AN ARTIFICIAL IMMUNE SYSTEM STRATEGY FOR ROBUST CHEMICAL SPECTRA CLASSIFICATION VIA DISTRIBUTED HETEROGENEOUS SENSORS

MASTERS THESIS

Mark A. Esslinger, Captain, USAF

AFIT/GCS/ENG/03-06

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GCS/ENG/03-06

AN ARTIFICIAL IMMUNE SYSTEM STRATEGY FOR ROBUST CHEMICAL SPECTRA CLASSIFICATION VIA DISTRIBUTED HETEROGENEOUS SENSORS

MASTERS THESIS

Presented to the Faculty of the

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science (Computer Science)

Mark A. Esslinger, BS

Captain, USAF

March 2003

AFIT/GCS/ENG/03-06

AN ARTIFICIAL IMMUNE SYSTEM STRATEGY FOR ROBUST CHEMICAL SPECTRA CLASSIFICATION VIA DISTRIBUTED HETEROGENEOUS SENSORS

Mark A. Esslinger, B.S.

Captain, USAF

Approved:

Professor Gary B. Lamont Thesis Advisor	12 MAR 03 Date
Professor Charles A Bleckmann Committee Member	12 Mar 03 Date
Dr. Robert L. Ewing Committee Member	12 MAR 03 Date
Dr. Gregg H. Gunsch Committee Member	12 MAR 03 Date

Acknowledgements

I would like to express my sincere appreciation to my thesis advisor, Dr. Gary B. Lamont, for his guidance and support throughout this course of study. His insight and experience was truly appreciated.

I am also indebted to my research sponsor, Dr. Robert Ewing, for providing direction, ideas, and behind the scenes support, that greatly contributed the success of this research effort.

Thanks also to all of the nameless professors who prepared me for this endeavor through a rigorous course of study and high expectations.

My deepest appreciation goes to my fellow students for providing the motivation and consolation to keep me focused on the goal.

Last, but definitely not least, I thank God for his divine guidance and for giving me the patience and perseverance to complete this monumental task.

Mark A. Esslinger

Table of Contents

Page

Page

List of Figures

List of Tables

Abstract

The timely detection and classification of chemical and biological agents in a wartime environment is a critical component of force protection in hostile areas. Moreover, the possibility of toxic agent use in heavily populated civilian areas has risen dramatically in recent months. This thesis effort proposes a strategy for identifying such agents via distributed sensors in an Artificial Immune System (AIS) network. The system may be used to complement "electronic" nose ("E-nose") research being conducted in part by the Air Force Research Laboratory Sensors Directorate. In addition, the proposed strategy may facilitate fulfillment of a recent mandate by the President of the United States to the Office of Homeland Defense for the provision of a system that protects civilian populations from chemical and biological agents. The proposed system is composed of networked sensors and nodes, communicating via wireless or wired connections. Measurements are continually taken via dispersed, redundant, and heterogeneous sensors strategically placed in high threat areas. These sensors continually measure and classify air or liquid samples, alerting personnel when toxic agents are detected. Detection is based upon the Biological Immune System (BIS) model of *antigens* and *antibodies*, and alerts are generated when an a measured sample is determined to be a valid toxic agent (*antigen*). Agent signatures (*antibodies*) are continually distributed throughout the system to adapt to changes in the environment or to new *antigens*. Antibody features are determined via data mining techniques in order to improve system performance and classification capabilities. Genetic algorithms (GAs) are a critical part of the process, namely in antibody generation and feature subset selection calculations. Demonstrated results validate the utility of the proposed distributed AIS model for robust chemical spectra recognition.

AN ARTIFICIAL IMMUNE SYSTEM STRATEGY FOR ROBUST CHEMICAL SPECTRA CLASSIFICATION VIA DISTRIBUTED HETEROGENEOUS SENSORS

I. INTRODUCTION

The Artificial Immune System (AIS) model has demonstrated aptitude in the classification of unknown elements within NP-Complete problem domains. The model presented draws its inspiration from the success of the AIS as applied to a wide range of problems such as intrusion detection [27] [16] [2], multimodal function optimization [20], and ecosystem management [48]. A framework for applying the AIS model is discussed that can quickly classify biological agents in a war or peacetime environment. The model is based upon a system of robust, scalable, efficient, and relatively simplistic sensors that can be "scattered" anywhere in a threat area and provide immediate warning the release of toxic agents.

Classification is determined chiefly via continuous analysis of chemical spectra by heterogenous sensors in an distributed AIS (DAIS) network. Operating as low-level agents in a hierarchical configuration [54], sensors continuously assess local environmental conditions and classify the resulting spectra as benign, naturally occurring elements (*self*), or harmful biological contaminants (*non-self*).

1.1 Overview

"Integrated bio-inspired circuits that sense, receive, transmit, and process signals are the *eyes, ears,* and *nose* of the millennium" [1]. Generically speaking, a sensor is any device that receives and responds to stimuli. Response does not imply any form of innate sensor "intelligence". Yet, by connecting multiple sensors and strategically placing them in threat areas, the utility of multiple sensor responses may

be enhanced. "Widely available Biological Agent detection and the integration of Chemical and Biological Agent detection into an embedded processor would greatly improve upon current fielded technology, better protect the warfighter, and considerably increase situational awareness by incorporating data obtained into current Command, Control, and Communication Systems" [25]. This application of biological principals to the information systems computational domain has been coined "Bioinformatics". As components of a DAIS, *stimuli* is provided via constant environmental measurements. As stimuli, measurements are determined to be *self* or *non-self* and evoke a *sensor response* proportional to a dynamic *affinity* "match" score. A match score is the determination of how closely a stimulus resembles *self* or *non-self*, while *affinity* refers to the threshold that must be reached in order to generate a biological response (*warning*) indicating that a *non-self* stimulus has been detected. On the surface, the process seems quite simple. However, "to process/store/analyze signals acquired from multiple physical sensors, hybrid systems with flexible and adaptable artificial intelligence are needed" [1]. Genetic Algorithms (GAs) provide the evolutionary ability to adapt to new environments and play a key role in the discovery of patterns to categorize signals in noisy environments. Figure 1 illustrates the complexity of interactions between multiple sensors and the need for "smart" algorithms to classify stimuli.

Real-time sensors have the potential to produce an extremely large amount of data about elements detected in the environment. These data must be classified as quickly as possible to provide adequate warning when chemical/biological elements are present. Figure 2 illustrates the process of "intelligent bio-inspired signal processing." Note the roles that pre-processing feature extraction and data fusion play in the decision-making loop.

1.1.1 Electronic Nose Research. Mammalian olfactory systems are capable of distinguishing between millions of different odors resulting in instantaneous recognition of multiple odor sources [10]. The mapping of biological principles in-

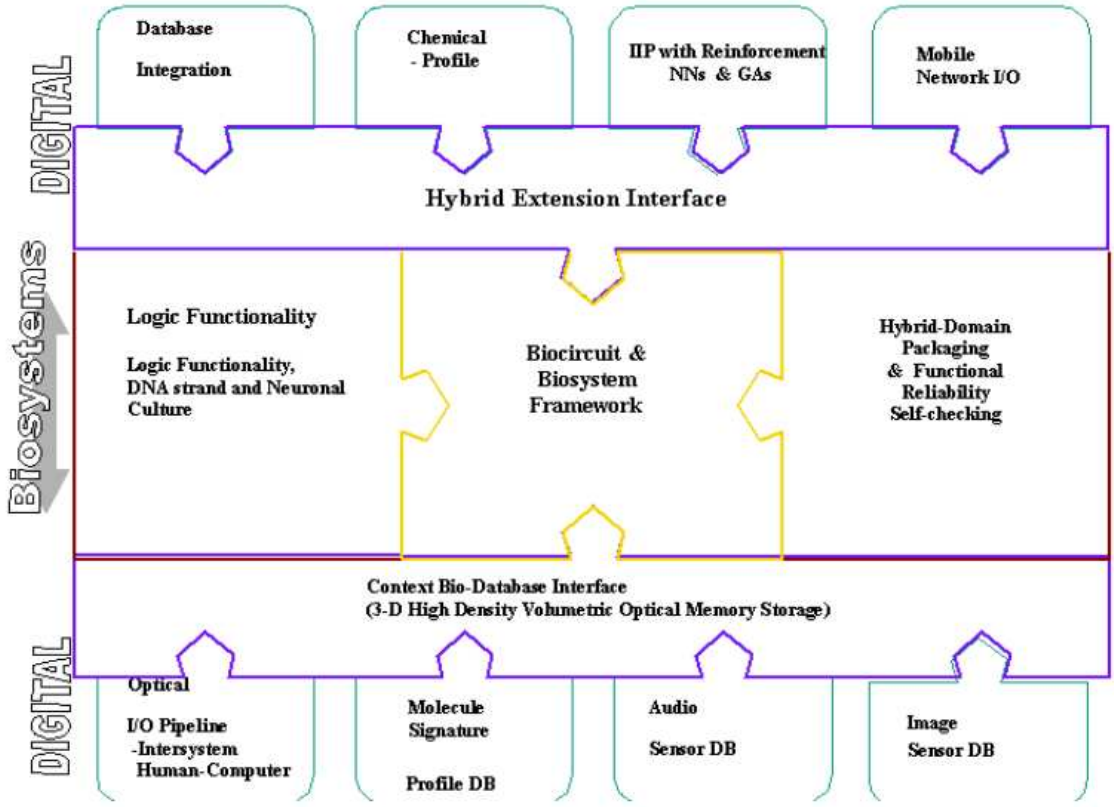

Figure 1 A Multidisciplinary Biotechnology System [1]

volved with odor recognition to the computational domain results in a system able to provide comparable, albeit limited, odor classification. Research in this "olfactory science" area is being undertaken by many government agencies and universities. Caltech's Microsystems Research Laboratory is conducting such research with the goal attaining an "understanding of biological olfaction and the construction of a silicon 'nose on a chip'" [10]. Applications of this technology include [10]:

- Chemical Analysis
- Environmental Monitoring
- Food Inspection
- Land Mine Detection
- Airport Luggage Inspection

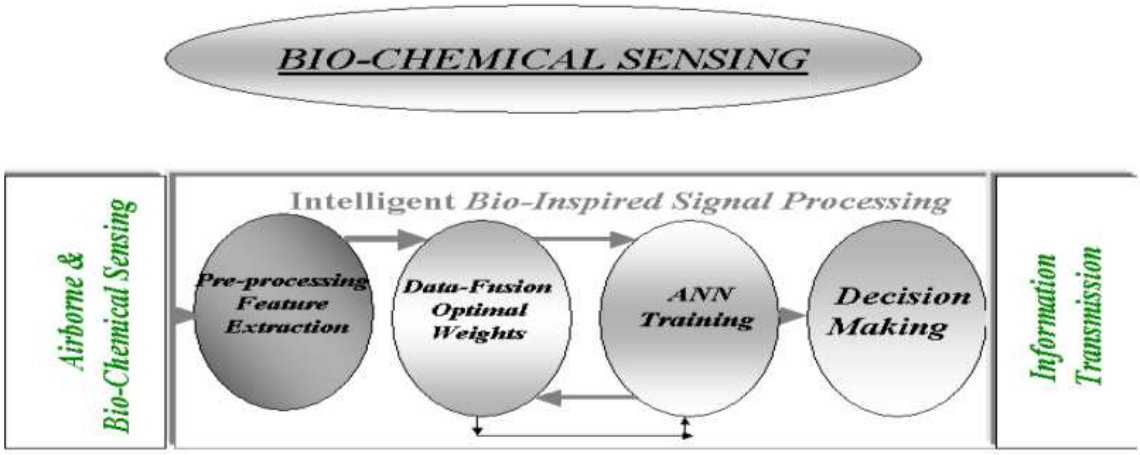

Figure 2 Bio-Inspired Signal Processing [1]

- Emission Control/Enforcement
- Narcotic Detection

E-nose realization involves three main research thrusts: (1) sensor technologies, (2) signal processing, and (3) classification methods. This thesis serves to complement the third area of study by providing an alternate and/or complementary odor classification method.

1.1.2 Sponsorship. General research sponsorship is provided by the Air Force Research Laboratory (AFRL), Sensors Directorate, under the guidance of Dr. Robert Ewing.

1.1.3 E-nose Biological Inspiration. As previously mentioned, biological inspiration for the e-nose is derived from mammalian olfactory systems. Biological olfactory systems consist of thousands of individual sensors located in the epithelium (10,000 in humans, 100,000 in dogs). At any given time, roughly 25% of these sensors are firing in response to stimuli. Sensors are tied to olfactory neurons that eventually transmit their signals to the olfactory cortex in the brain. Presented with these signals, the brain, then performs odor classification and recognition [10]. Research at Caltech currently models this function of the olfactory cortex via an artificial

neural network; however, an AIS could be used to complement the classification process.

1.1.4 Feature Extraction and Feature Subset Selection. As illustrated in Figure 2, stimuli feature extraction is a key component in signal processing. The number of features that can be extracted by current sensors is limited only by the complexity of the given hardware. Each feature extracted directly increases the dimensionality of the classification process, providing additional information about the stimuli. The extraction of features that can be used to efficiently represent and classify stimuli is critical to the success of the DAIS. The goal is to determine the smallest feature subset necessary for accurate classification. Most chemical sensors produce a Raman spectra plot, based upon the reactance of the chemical to different wavelengths of the electromagnetic spectrum resulting in an intensity vs. wavelength plot. There are theoretically an infinite number of features that may be extracted from such a plot. In addition, independent variables such as temperature, humidity, wind direction, and chemical intensity may be included in the resulting chemical feature set. Given these many variables, the Genetic Rule and Classifier Construction Environment (GRaCCE) program presented in [56] and parallelized in [39] (as pGRaCCE) is used to determine the best feature subset for classification and use within the DAIS.

1.2 Research Goals and Objectives

The increasing threat of biological warfare facing today's military forces is an area of concern for all military members. As such, maintaining situational awareness of environmental conditions is the first step in preventing a successful biological attack. The research *goal* is to develop a computational framework for a distributed sensor network capable of providing early warning in the event of a chemical/biological attack. In order to accomplish this goal, the following three objectives are addressed:

1. Analyze the performance of pGRaCCE on a real-world data set
2. Analyze the performance of a parallel implementation of J. J. Grefenstette's genetic algorithm program Genesis [36] and its ability to evolve antibodies capable of classifying multiple variations of real-world toxic chemicals
3. Design,implement, and test a basic DAIS that models a real-world network of sensors capable of classifying chemical spectra and producing warnings when *non-self* chemicals are present

1.3 Approach

The phased approach taken focuses on feature subset selection, the evolution of antibodies, and the development of a representative DAIS that uses them. Design, analysis, and testing takes place in three phases:

Phase I: Analyze the performance of pGRaCCE against multiple real-world data sets

Phase II: Analyze the ability of parallelized Genesis to evolve antibodies capable of classifying a given subset of the real-world data used in Phase I, given the feature subsets produced in Phase I.

Phase III: Design, implement, test, and evaluate the performance of a DAIS capable of producing warnings when the items from the data subsets in Phase II are present.

1.4 Software Design Process

Design and implementation of software systems necessitates a procedure that takes a top-down approach that starts with the problem statement and ends with a fully implemented system. This process requires iterative application of the following five steps until the system performs as designed [53]:

1. Define/analyze problem domain requirements, including partial operational specification over input and output domains. Use symbolic notation whenever possible to simplify transition between steps.
2. Choose an algorithm domain specification strategy based upon known models in current research.
3. Evolve a general solution design specification (algorithmic, iterative, or recursive) and an operational design specification using algebraic or symbolic notation. Extend notation specified in previous step. Specialize the algorithm template with the problem domain.
    - Instantiate problem design specification within selected algorithmic method through problem domain data structures
    - Algorithm design templates and design specifications are developed and imported to support the top-down design process
4. Refine solution design recursively to low-level design by incorporation additional data structures and operations as required to create a refined algorithmic design template.
5. Map low-level design to selected (compiler) language and reusable components.

1.5 Assumptions

In order to reasonably limit the scope of discussion, it is assumed the reader has a general knowledge of the following subjects:

1. Computer Engineering and Computer Science, to include: parallel and distributed computing, evolutionary computing, computer architectures, computer operating systems, computer programming, general algorithms and complexity.
2. Probability and statistics

3. Basic biological concepts, including: immunological functions, vaccination, DNA replication and operations

1.6 Risks and Concerns

The largest risk to success of this research effort is the broad scope of subjects addressed. In order to focus research efforts on relevant issues, only the high interest topics are addressed.

1.7 Thesis Outline

This thesis consists of seven chapters. This chapter provides a basic introduction to the thesis research topic, to include an overview of the problem domain, research goals, associated objectives, assumptions, risks, and overall layout of the thesis. Chapter 2 focuses on historical perspective, problem domain models, and possible algorithm domains for the solution, statistical techniques, and software engineering approaches. Chapter 3 presents a high-level design of the systems in question and maps the problem domain to appropriate structures. Chapter 4 discusses the low-level implementation details of the system. Chapter 5 gives a detailed justification of the experimental design process and presents the overall design of experiments. Chapter 6 presents the results and an analysis of experiments. Finally, chapter 7 presents conclusions and recommendations derived from the research effort.

II. BACKGROUND

This chapter presents supplemental background knowledge to enhance the development of the DAIS. Due to the broad scope of disciplines discussed, the relevant characteristics of genetic algorithms, artificial immune systems, parallel computing, sensors, and data mining, are presented. Each section is preceded by a brief history of previous associated research in each respective area.

2.1 Genetic Algorithms

Genetic algorithms provide the evolutionary ability to improve DAIS performance and classification ability. One of the first descriptions of the use of an evolutionary processes for computer problem solving appeared in articles by Friedberg in 1958 [32] and 1959 [33]. "This work represented some of the early work in machine learning and described the use of an evolutionary algorithm for *automatic programming*, i.e. the task of finding a program that calculates a given input-output function" [21]. Many studies sprung from this paper and others by Bremermann in 1962 [9], Box in 1957 [7], and Box et. al in 1969 [8]. As is the case with many ground-breaking research ideas, these early studies were reviewed with skepticism. However, by the mid-1960's the bases for the three main focuses of evolutionary computation were clearly established [21]. These three main focuses were: Evolutionary Programming (EP), Evolutionary Strategies (ES), and Genetic Algorithms (GAs). GAs are used exclusively as a process for search space exploration and exploitation (E & E), and are therefore examined in detail. Further details concerning EP and ES are in Appendix A-1.

GAs were first conceptualized by Holland in many of his papers written in the early 1960's (e.g. see [45]). Holland set out to understand the underlying principles of adaptive systems–systems capable of responding to interactions with their environment through self-modification. By the mid-1960's, Holland's ideas began

to take computational form in thesis work of several of his PhD students. The distinctive feature of these theses was the successful use of competition and innovation to provide the ability to dynamically respond to unanticipated events and changing environments.

2.1.1 Properties. All basic instances of GAs share a number of common properties [4]:

- All instances utilize the collective learning process of a population of individuals. Each individual represents a search point in the space of potential solutions to a given problem.
- Individuals are used to generate descendant individuals via a randomized process that models organic mutation (subsection 2.1.3.2) and crossover/recombination (subsection 2.1.3.3).
- A measure of quality, or *fitness* is assigned to individuals in order to improve the likelihood of choosing (*selection*, subsection 2.1.3.1) quality individuals for reproduction and transference to the next generation. Highly fit individuals are more likely to reproduce than individuals that are relatively worse in fitness.

2.1.2 Representation. There are many way to represent individuals within the search space. Representation typically mirrors the solution space as closely as possible in order to simplify execution. Real-valued, integer-valued, and binary vectors are commonly used in this process. Individual vector sizes vary based the dimensionality, or number of decision variables within the search space. For instance, a four-featured binary individual would be used to represent a binary search space in four dimensions with the range of possible values of "`0000`" to "`1111`".

Individual structures are often referred to as *chromosomes*, they are the *genotypes* that are manipulated by the GA. If individuals are represented by binary strings (as above), the value of each locus on the bitstring is referred to as an *al-*

lele. Sometimes the values of each loci are called *genes*; while other times genes are combinations of alleles that have some phenotypical meaning, such as parameters [24].

2.1.3 Operators. The genetic operators, *selection, mutation,* and *crossover*, are central to GA execution and serve to distinguish them from other evolutionary computation techniques. Each operator is discussed in greater detail in the following subsections.

2.1.3.1 Selection. "The primary objective of the selection operator is to *emphasize* better solutions in a population" [23]. In short, the selection operator determines which chromosomes continue on to the next generation. All selection techniques (except *random selection*) depend upon some measure of relative fitness for each chromosome. The central idea is that individuals that are more highly fit have a higher probability of selection. Symbolically, the basic selection operator can be represented by the following pseudocode. The fitness function is represented by $\mathbf{F}(t)$ [23].

Input: μ : parent solutions
λ : offspring solutions
q : selection pressure parameter
$P(t) \in I^{\mu}$: population at iteration t
$P'(t) \in I^{\lambda}$: offspring population at t to be carried on to iteration $t+1$

Output: $P''(t) = \{a''_1, a''_2, ..., a''_{\mu}\} \in I^{\mu}$

1. **for** $i \leftarrow 1$ **to** μ
   $a''(t) = \{a''_i(t) \leftarrow s_{selection}(P(t), P'(t), \mathbf{F}(t), q);$
2. **return**$(\{a''_1, a''_2, ..., a''_{\mu}\} \in I^{\mu});$

The main types of selection operators include:

Proportional Selection: the expected number of copies a solution receives is assigned proportionally to its fitness. Thus, a solution having twice the fitness of another solution receives twice as many copies. This type of selection is also known as *roulette wheel* selection, because if the population resided on a roulette wheel, each individual would occupy an area proportional to its fitness. Then, the roulette wheel is spun as many times as the population size and individuals are selection based upon the result of each spin [23]. This can result in scaling problems if a population contains a solution with *exceptionally* better fitness than the rest of the population. This "*supersolution*" occupies most of the roulette wheel area, resulting in convergence to a possibly suboptimal solution in the supersolution region of the search space. More specifics concerning the many variations of proportional selection can be found in [37] and [5]

Tournament Selection: the scaling problem discussed above is eliminated by playing "tournaments" among a specified number of individuals according to their fitness functions. For example, in a three-way tournament, three individuals are deterministically or randomly chosen from the parent population. The individual with the highest fitness among the three is selected. See [6] for a detailed analysis of this selection type.

Rank Selection: similar to proportional selection, except that solutions are ranked according to descending or ascending fitness. Thereafter, individuals are selected according to their ranked fitness value. There are a number of different schemes that are based on the ranking concept, see [55].

Boltzmann Selection: a modified fitness is assigned to each solution based on a *Boltzmann probability distribution* (eq. 1):

$$F_i = \frac{1}{1 + exp(\frac{F_i}{T})} \tag{1}$$

where T is a parameter analogous to the temperature term in the Boltzmann distribution. T is reduced by a predefined measure during each iteration. Since T is initially large, solutions are all just as likely to be selected; but, as the number of iterations increases, T decreases and only good solutions have a high likelihood of being selected.

2.1.3.2 Mutation. The mutation operator models the erroneous replication of individuals that sometimes takes place during biological reproduction. Typically, small errors are introduced to reduce the likelihood of moving individuals to drastically different parts of the search space.

2.1.3.3 Crossover/Recombination. Crossover and recombination are different terms that represent the same basic concept: the exchange of information between two or more existing individuals. For example, consider the following binary string: 10100011101011. Also, assume that the following is also a binary string, where x = 1 and y = 0: xyxyyxxxyxyxyx. A one-point crossover after the 5th allele value would look like [73]:

10100∨011101011

xyxyy∨xxxyxyxyx

and after swapping the segments, the resulting offspring would become:

xyxyy011101011 and 10100xxxyxyxyx

2.1.4 Basic GA Algorithm. All basic instances of GAs follow the same algorithmic template. In [4], Bäck defined a symbolic framework for defining GA

operators and components. Using this framework, I denotes an arbitrary space of individuals $a \in I$ and $F : I \rightarrow \Re$ to denote a real-valued fitness function of individuals. Using μ and λ to denote parent and offspring population sizes where $P(t) = (a_1(t), ..., a_\mu(t)) \in I^u$ characterizes a populations at generation t. Selection, mutation, and recombination operators, defined as operators s, m, and r transform complete populations of individuals over n generations. Formally, these operators are defined as [4]:

$$s : I^\lambda \rightarrow I^\mu$$
$$m : I^K \rightarrow I^\lambda$$
$$r : I^\mu \rightarrow I^K$$

These operators typically depend upon additional parameters Θ_s, Θ_m, and Θ_r, that are characteristic for each operator and the representation individuals.

Additionally, an initialization procedure, ι, generates the first population of individuals at time $t = 0$ and evaluates the fitness of each individual. Individuals are typically initialized in some random fashion in order to start with a population evenly distributed over the search space; however, individuals may also be initialized *a priori* in search space locations known to contain individuals with high fitness levels. Finally, termination criterion is established to determine when or if the algorithm should stop. The termination criterion typically stops the algorithm after a specified number of generations, when relative population fitness scores have not improved by a specified percentage over a specified number of generations, when a desired fitness score has been achieved, or any combination of the these.

Having defined the basic components of the GA, they may be combined in a simple recombination-mutation-selection loop as follows [4] and illustrated in Figure 3:

Input: $\mu, \lambda, \Theta_\iota, \Theta_r, \Theta_m, \Theta_s$

Output: a^*, the best individual found during the run, or

P^*, the best population found during the run.

1. $t \leftarrow 0$;
2. $P(t) \leftarrow \text{initialize}(\mu)$
3. **while** $(\iota(P(t), \Theta_\iota) \neq \text{true})$ **do**
4. $P'(t) \leftarrow \text{recombine}\ (P(t), \Theta_r)$;
5. $P''(t) \leftarrow \text{mutate}(P'(t), \Theta_m)$;
6. $\mathbf{F}(t) \leftarrow \text{evaluate}(P''(t), \lambda)$;
7. $P(t+1) \leftarrow \text{select}(P''(t), F(t), \mu, \Theta_s)$;
8. $t \leftarrow t + 1$;
9. **od**

A description of each line follows:

Line 1: Set starting time $t = 0$

Line 2: Initialize the parent population μ

Line 3: Enter **while** loop, with termination criterion ι according to termination parameters Θ_ι

Line 4: Recombine current population according to parameters Θ_r

Line 5: Mutate recombined population according to parameters Θ_m

Line 6: Evaluate mutated population to determine fitness $\mathbf{F}(t)$

Line 7: Select μ individuals from mutated population according to their fitness values and selection parameters Θ_s

Line 8: Increment the current time by one

Line 9: Return to **Line 3**

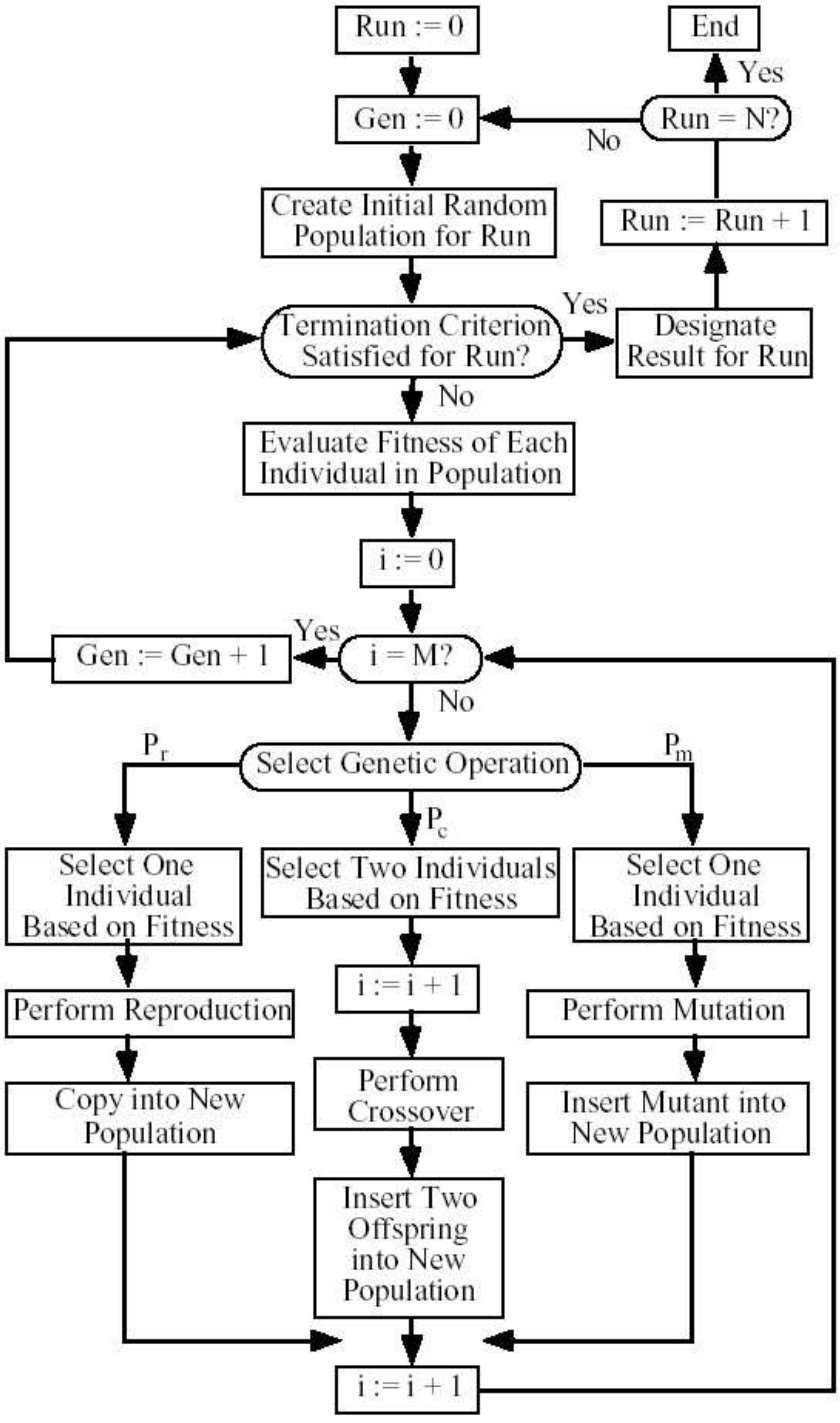

Figure 3 Flowchart for the Conventional GA [51]

2.2 Artificial Immune Systems

The AIS as a method of classification has been studied by many researchers. Parallels between immunology and classifier systems were noted by [26] as early as 1986. More recently, Forrest [27, 28], Dasgupta [18, 16] , De Jong [22], Lamont [54], and many others have expanded upon the topic, resulting in hundreds of publications and international conferences on the subject. Given the diversity of immune system concepts available for exploitation, the many possible applications include computer security [27], virus detection [30], UNIX process monitoring [29], anomaly detection in time series data [19], fault diagnosis [49], and chemical spectra recognition [18]. The last application (chemical spectra recognition) is the focus of this research. Discussion of the AIS begins with a summary of the biological immune system and is followed by its application to AIS computing concepts.

2.2.1 Biological Immune System. The biological immune system (BIS) defends the body against harmful diseases and infections. It is capable of recognizing virtually any foreign cell and destroying it. In order to do this, the BIS must distinguish between molecules and cells that belong to the body and those that do not. This concept of *self* from the dangerous *non-self* is the basis of all immune system operations. The exact possible number of foreign body invaders is unknown, but it has been estimated to be in excess of 10^{16} [47]. These foreign proteins must be distinguished from an estimated 10^5 different proteins of self, and recognition must be highly specific [67].

The architecture of the biological immune system is multi-layered, with defenses at many levels. The first and outermost layer is the skin. A second barrier is the physiological, where temperature and pH provide inappropriate living conditions for most foreign invaders (pathogens). After pathogens have circumvented these first two layers of defense, they must battle with the third and final layer, the innate immune system and adaptive immune response. The innate immune system

uses macrophages to ingest extracellular debris and clear the system of other foreign invaders. Adaptive immune response is the most complicated defense mechanism. Response is "adaptive" in that it is capable of identifying and eradicating pathogens that have previously never been encountered. This requires interactions between many different types of cells and molecules [67]. Figure 4 demonstrates the layered immunological responses in the BIS. The adaptive immune system consists primar-

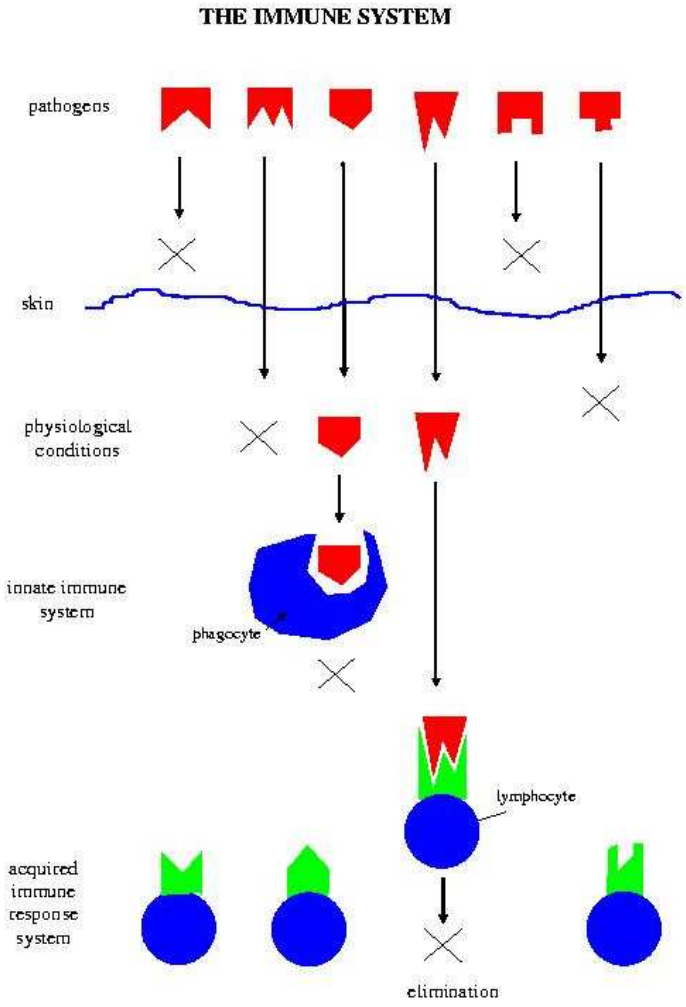

Figure 4 Layered Immunological Response [43]

ily of white blood cells, called *lymphocytes*. Lymphocytes circulate throughout the

body and identify molecules that exhibit non-self patterns while ignoring molecules that resemble self. For this reason, lymphocytes are considered *negative* detectors. Detection and recognition of non-self occurs when lymphocyte receptors bond with pathogen receptors that cover the surface of each molecule. The more closely the receptors on each molecule match, the higher the electrostatic bond between them (or, the higher the *affinity*). All detection is *approximate*; that is, individual lymphocytes bond to several different kinds of structurally related pathogens with a certain affinity [67]. Figure 5 illustrates the process of detection between complementary antigens and detectors. Note that some detectors do not form a chemical bond due to structural differences, while detectors with similar structures bond readily.

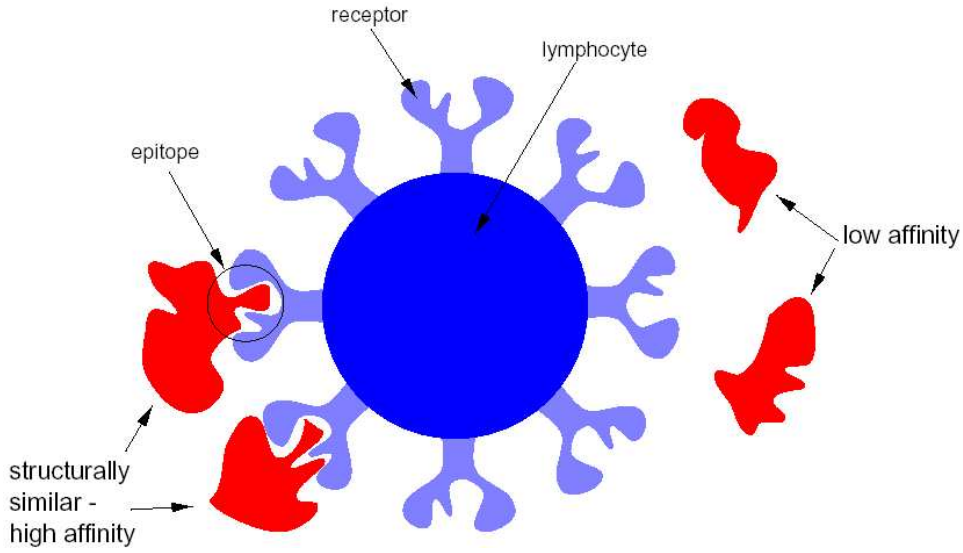

Figure 5 The Detection Process as a Function of Detector Affinity[42]

The ability to detect most pathogens requires a large diversity of lymphocyte receptors. These receptors are generated through a genetic process that introduces a huge amount of randomness. Given this random process, the potential exists for creation of lymphocytes that detect self. Lymphocytes that detect self are eliminated through a process called *clonal deletion* which takes place in the thymus. Almost all self-proteins in the biological body repeatedly pass through the thymus. Any

developing lymphocytes that bond to self-cells are eliminated before introducing them into the body [67].

There are never enough lymphocytes in the body to provide complete coverage of all possible pathogens. The immune systems has several mechanisms in place to mitigate this issue. These mechanisms make the immune system more dynamic and specific, improving classification and adaptation capabilities. The process is made dynamic through the continual circulation of short-lived lymphocytes throughout the body. The total lymphocyte population then turns-over every few days, replaced by younger randomly generated lymphocytes. This improves the immune system's ability to protect against diverse pathogens over a longer period of time; the longer a pathogen is in the body, the more likely it is to be detected by a wide array of lymphocytes [67].

Immune specificity is provided by an established immune learning and memory. When a pathogen that has never been encountered is detected, the immune system "learns" the structure of this specific pathogen and triggers a response that evolves a set of lymphocytes with a high affinity for that pathogen (called *affinity maturation*). These high-affinity lymphocytes are stimulated to reproduce in great numbers, and the resulting lymphocytes have a large number of mutations, effectively protecting the body from variants of the detected pathogen. Speed of response to previously encountered pathogens is then improved due to an acquired *immune memory* consisting of previously adapted lymphocytes [67].

2.2.1.1 BIS Characteristics. The key features of the biological immune system which provide several important aspects to the field of information processing may be summarized under the following terms of computation [17]:

Recognition: the immune system can recognize and classify different patterns and generate selective responses. Recognition is achieved by inter-cellular binding– the extent of this binding is determined by molecular shape and electrostatic

charge. Self/non-self discrimination is one of the main tasks the immune system solves during the process of recognition.

Feature Extraction: Antigen Presenting Cells (APCs) interpret the antigenic context and extract the features, by processing and presenting antigenic peptides on its surface. Each APC serves as a filter and a lens: a filter that destroys molecular noise, and a lens that focuses the attention of the lymphocyte - receptors.

Diversity: the BIS uses combinatorics (partly by a genetic process) for generating a diverse set of lymphocyte receptors to ensure that at least some lymphocytes can bind to any given (known or unknown) antigen.

Learning: the BIS "learns", by experience, the structure of a specific antigen. The system makes changes in lymphocyte concentration via *clonal expansion* during the primary response (the first encounter of the antigen).

Memory: when lymphocytes are activated, a few of each kind become special memory cells which are content-addressable. The longevity of these cells is an inherent mechanism of the dynamic process and requires continued stimulation by residual antigens. The system maintains an ideal balance between economy and performance by maintaining minimal, but sufficient, memory of the past.

Distributed Detection: the immune system is inherently distributed. Lymphocytes circulate throughout the body and organs and encounter various antigens, stimulating specific immune responses.

Self-regulation: the mechanisms of immune response are not controlled by any one central organ and can be either local or systemic, depending on the route and property of the antigenic challenge.

Threshold Mechanism: immune response takes place only above a certain matching threshold, related to the strength of chemical binding.

Co-stimulation: regulates the activation of B-cells, while a second signal (from helper T-cells) ensures tolerance and distinguishes between harmful invaders or false alarm.

Dynamic Protection: clonal expansion and somatic hyper-mutation allow generation of high-affinity immune cells (called affinity maturation). This process balances exploration versus exploitation (E & E) in adaptive immunity and increases the coverage provided by the immune system over time.

Probabilistic Detection: the cross reaction in immune response is a stochastic process, where detection is approximate. Lymphocytes can bind with several different kinds of structurally related antigens.

To summarize, the biological immune system has many features that are desirable from the standpoint of computer science. The BIS is massively parallel and the its functions are truly distributed. Each component is individually disposable, yet the system as a whole is still robust. Previously detected infections are eliminated quickly, while new or novel infections illicit an autonomous response that improves classification capability and overall system performance [67].

2.2.2 Mapping the BIS to the AIS. Artificial immune systems are a computational instantiation of the biological characteristics of the BIS described in section 2.2.1.1. To be thorough, there exists a one-to-one mapping for each BIS characteristic as illustrated below:

Recognition: AIS's recognize and classify different patterns and generate selective responses (or warnings, as presented here) . Recognition is achieved by detectors–the extent of which is a result of a specified detector matching function. This results in self/non-self discrimination when detectors are presented with sensory input.

Feature Extraction: Detectors focus attention on the features of system activity that represent non-self by continually improving classification ability.

Diversity: the AIS uses computational combinatorics to generate a diverse set detectors to ensure that known and unknown antigens can be classified.

Learning: the AIS improves classification capabilities through exposure to items representative of self and non-self. The system makes changes in detector concentration via *clonal expansion* and improves individual detectors via *affinity maturation*.

Memory: when detectors are activated, a few become special memory cells that remain in the system for a specified period of time. The longevity of detectors is a mechanism of the dynamic AIS process and requires continued stimulation by non-self. This results in an balance between economy and performance by maintaining minimal, but sufficient, memory of non-self.

Distributed Detection: the AIS inherently distributed by design. Detectors at different nodes come into contact with non-self, stimulating specific immune responses throughout the entire system.

Self-regulation: AIS operations are not controlled by any one central node.

Threshold Mechanism: Detector activation takes place only above a specified matching threshold, related to the strength of the match between the detector and non-self.

Co-stimulation: regulates the activation of detectors, based upon the activation of other detectors to similar antigens.

Dynamic Protection: clonal expansion and somatic hyper-mutation allow generation of high-affinity detectors. This balances E & E, increasing the coverage provided by detectors over time.

Probabilistic Detection: Non-self detection is approximate, a product of the generality or specificity of the chosen matching function.

2.2.3 AIS Characteristics. The overarching goal of any AIS is self/non-self discrimination and classification. All discrimination between self and non-self in the BIS is based upon chemical bonds that form between protein chains. In the AIS, to preserve generality, protein chains are modelled as binary strings of fixed length. String length is dependent on the number of features chosen to represent each individual or chromosome [44]. Symbolically, if the set of all strings of length l forms a universe of strings, U, then this universe may be split into two disjoint subsets S and N. S is the set of all strings that represent *self*, and N is the set of all strings that represent *non-self*. In other words, the universe of possible strings may be represented by [44]:

$$U = S \bigcup N, S \bigcap N = \emptyset$$

Given an arbitrary string from U, the AIS must then determine whether it belongs to S or N. The AIS then faces two basic discrimination errors: *false positives*, and *false negatives*. "A false positive occurs when a self string is classified as anomalous, and a false negative occurs when a non-self string is classified as normal" [44]. The BIS also makes similar errors: a false negative occurs when the IS fails to detect and fight off pathogens, and a false positive error occurs when the BIS attacks the body (known as an autoimmune response). In the body, both kinds of errors are harmful, so the BIS has apparently evolved to minimize those errors; similarly, the goal of the AIS is to minimize both kinds of errors [44]. Figure 6 illustrates this as a two-dimensional representation of a universe of strings. If a point lies within the shaded area, it is self, otherwise it is non-self. The AIS detection system is illustrated by the black outline around self space. As shown by the overlap between self and non-self with the detection line, it is evident that the detection system fails to properly categorize some strings, resulting in false positives and false negatives. A properly tuned AIS adjusts the detection line as to minimize this overlap [44].

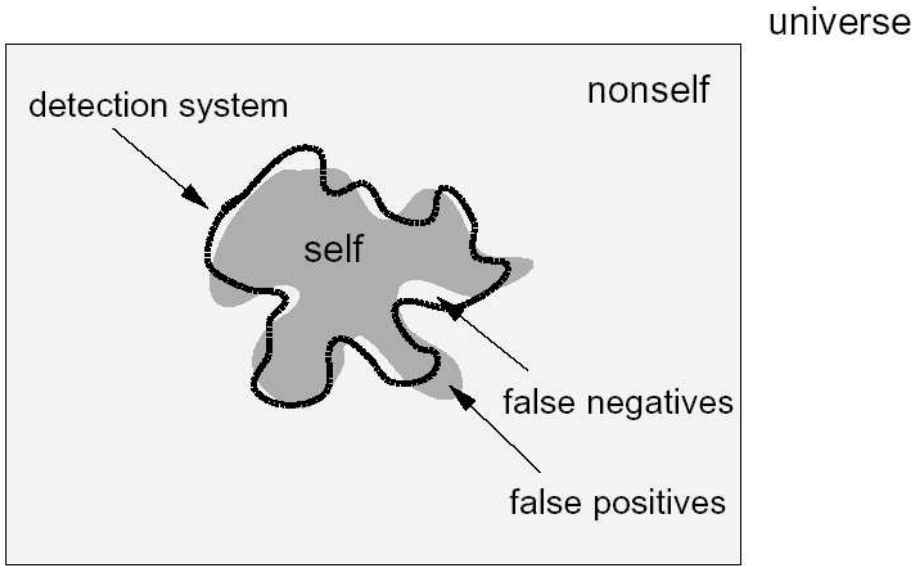

Figure 6 Graphical Depiction of the Universe of Strings [44]

2.2.4 AIS Operators. AIS operators model themselves after the BIS model to simulate an environment capable of self/non-self recognition and classification. These operators include:

- negative selection
- imperfect matching function
- affinity maturation
- dynamic clonal selection
- costimulation

Together, and with good variable selection, these concepts may be used to improve the efficiency of the GA, and therefore the efficiency of the AIS. Each is described in more detail in the following subsections.

2.2.5 Negative Selection. Within the constructs of the DAIS, negative selection is the process of detecting when evolved detectors may actually match benign

measurements produced by sensors. This prevents bad detectors from being introduced into the system, thereby producing erroneous detection results. The basic idea is to "use a immunological inspired *negative selection* algorithm to generate non-self samples (antigens). Then apply a classification algorithm to generate the characteristic function of the self (or non-self). This characteristic function correspond *[sic]* to the anomaly detection function" [35].

"Each detector is created with a randomly-generated bit string (analogous to a receptor), and remains immature for a time period T. called the *tolerisation* period. During this time period, the detector is exposed to the environment (self and possibly non-self strings), and if it matches any bit string it is eliminated. If it does not match during the tolerisation period, it becomes a mature detector (analogous to a naive B-cell). Mature detectors need to exceed the match threshold in order to become activated, and when activated they are not eliminated, but signal that an anomaly has been detected. Clearly, the assumption here is that if a circulating immature detector matches some self string, it encounters, with high probability, that self string during its tolerisation period, whereas immature detectors that match non-self strings encounter, with low probability, those non-self strings during their tolerisation period" [44]. Figure 7 illustrates detector generation via negative selection. If detectors (represented by dark circles) match any string in the self set within the tolerisation period, they are eliminated and randomly regenerated; otherwise, they are introduced into the system until a sufficient number of detectors are cover the space of non-self strings.

The negative selection algorithm is described in Gonzalez et. al. [35] and consists of three basic steps:

1. Define self as a collection of strings S of length l over a finite alphabet. In the AIS model, S may be generated by taking measurements of known clean air/liquid samples.

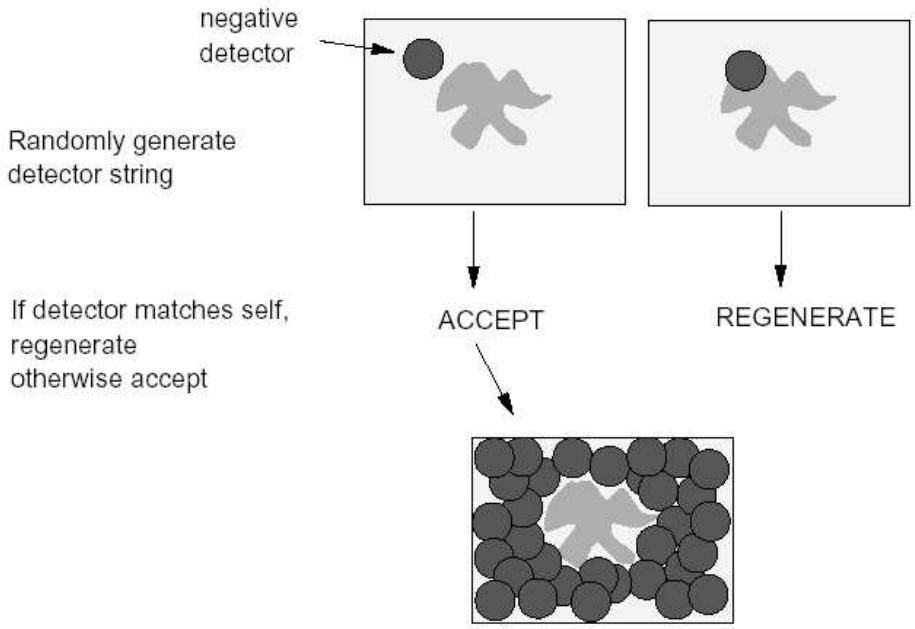

Figure 7 Detector Generation via Negative Selection [44]

2. Generate a random set of detectors D, such that $D \neq S$
3. Monitor S for changes by continually matching the detectors in D to S. If a change has occurred, there is be a match as the detectors in D are designed to not match any of the original strings in S.
4. Repeat

Figure 8 illustrates the flow of the negative selection algorithm.

2.2.6 Imperfect Matching Function. "As in the biological immune system, detection of an antigen...is accomplished by 'binding' the antibody to it via some imperfect matching process" [2]. Of course, whether or not a match is determined in the AIS depends entirely on the choice of the matching rule chosen. This should be accomplished via a matching function with an *affinity threshold* that varies with the

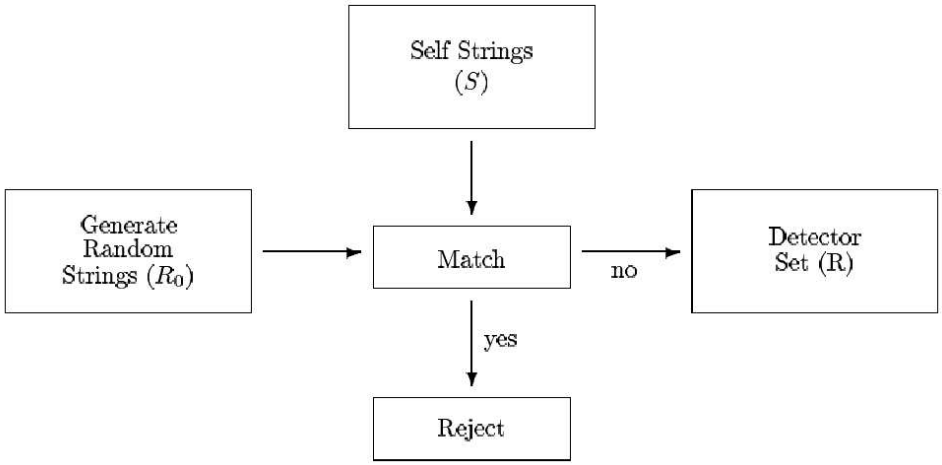

Figure 8 Flowchart of the Negative Selection Algorithm [30]

relative strength of the detector. Detectors that match more antigens have a higher fitness level, and therefore a lower affinity threshold for detection.

2.2.7 Affinity Maturation. The process of affinity maturation can be used as search for better detectors. "The goal of affinity maturation is to maximize the hypervolume defined by the ranges of the antibody's features; a genetic algorithm-based search is used for this process" [2]. The process proceeds by "hypermutating" (mutation with a high probability) newly created detectors to allow the detectors to improve their affinities with recognized antigens, while the rate of mutation is inversely proportional to the affinity of the parent cell with the recognized pattern [15].

2.2.8 Dynamic Clonal Selection. Kim [50] coined the term *dynamic clonal selection* (DCS) as an AIS which has the following two properties: (1) the AIS learns new behaviors by exposure to a small subset of antigens at one time, and (2) its detectors should be replaced whenever previously observed normal behaviors no longer represent current normal behaviors. DCS achieves population adaptability via "coordinated dynamics of three different detector populations: immature, mature,

and memory detector populations". The specifics of the algorithm are noted in Kim et. al. [50]; however, a more general AIS algorithm is specified as [15]:

```
Createapopulation of k antibodies (feasible solutions)
For each generation, do
  For each antibody, do
    decode the antibody
    determine the antibody affinity
  determine the number of clones of each antibody
  determine the number of mutations
  do cloning and mutation
  For Each clone, do
    decode the clone
    determine the clone affinity
    if afin(clone) > afin(antibody) → antibody = clone
While stopping criterion = false
```

2.2.9 Costimulation. Because the DAIS describes defines non-self as anything outside of the self-training data, the space has to be well-defined in order for it to work effectively. "However, since self is not perfectly defined and may drift over time, the system is likely to produce false positives" [2]. "Costimulation" is the process of trying to reduce the number of false-positives with the trade-off of possibly increasing the false-negative rate. In this process, any detection is compared to the results of other sensors to see if they would also determine the measurement to be an antigen. If yes, the measurement is marked as a biological agent, and warning is produced; if not, no warning is produced. So, warnings are not produced unless more than one sensor also determines that a biological attack is in progress. In the

proposed AIS, costimulation would likely be the responsibility of the network node N, in order to reduce the workload on each individual sensor.

2.2.10 Alternate Approaches. Many different approaches have been presented in literature detailing methods of antigen/antibody representation and operation. For completeness, two are presented.

2.2.10.1 The De Jong/Spears and Forrest Approach. De Jong and Spears' definition of "the traditional internal representation used by GAs [in pattern recognition] involves using fixed-length (generally binary) strings to represent points in the space to be searched" (i.e. the search for acetone and methanol in spectrum analysis plots) [22]. This definition can be applied when categorizing spectra by analyzing the spectra encoded as binary files. However, in a broader sense, not all spectra may be evident through simple examination of raw binary data by a classifier system. Further, this representation does "not appear well-suited for representing the space of concept descriptions which are generally symbolic in nature, which have both syntactic and semantic constraints, and which can be of widely varying length and complexity." [22]

The Forrest approach models the BIS by applying GAs to the evolution of a population of antibodies to detect antigens [i.e. spectra]. The goal is to evolve an antibody population capable of classifying more than one antigen, thereby becoming "generalists" and recognizing antigen class boundaries. Antigens and antibodies are represented by binary strings of length l (64-bits, in the Forrest examples). It should be noted that this approach is "interested [only] in the recognition properties of the immune system[and therefore] does not consider how the immune system neutralizes an antigen once it is recognized." [31]. An antibody "is said to match an antigen if their bit strings are complementary. Since each antibody must match against several different antigens simultaneously," perfect bit-wise matching is not required [31]. The match function M_0 simply counts the total number of bits that

differ between antigen and antibody and is represented by the M_o function (eq. 11). "Antibodies are matched against antigens, scored according to the fitness function M_o, and evolved using a conventional GA." [31] Figure 9 illustrates the basic BIS model: GA execution continues until each antigen is represented by at least one antibody and no self-strings are recognized.

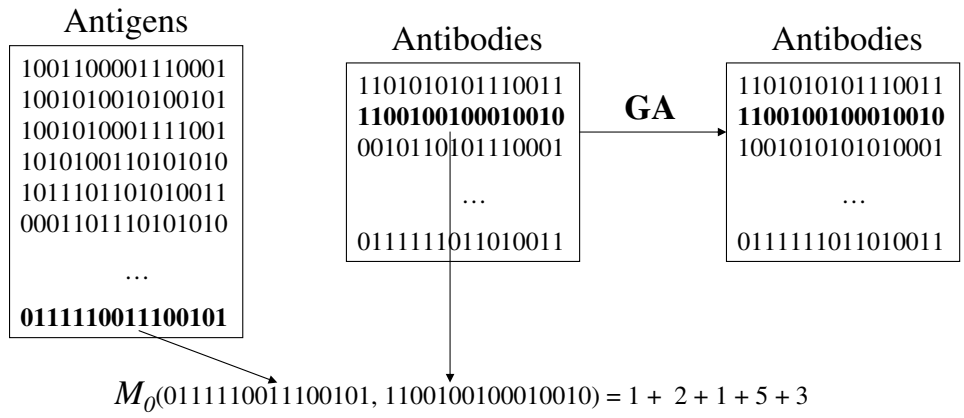

Figure 9 Graphical Depiction of GA Matching Function

Strings found to match self-strings are eliminated via "negative selection"[7]. Antibody strings are assigned higher fitness values when a greater number of matches occur. The fitness of each antibody *(*j)j is calculated by the following procedure [31]:

1. Choose a sample ρ antigens randomly from the initial set of antigens (with replacement). Note that the total antigen population from which this sample is chosen remains fixed throughout the run of the GA.
2. For each antigen k in the sample, compute the match score *M(k,j).*
3. The fitness of antibody j is the average match score computed over the sample of ρ antigens.

Using this model, Forrest et. al. recognized that as the population of antigens evolves to increased levels of generality, it becomes difficult to maintain enough diversity to recognize different classes of new antigens. That is, "for each population size there is a maximum 'carrying capacity'" [31]. However, this problem can

be resolved by increasing the population of antibodies and the number of antigens sampled per GA cycle. In general, there must be at least 15 antibodies for every antigen "type" (or class).

As stated earlier, adaptive pattern recognition of class boundaries involves both concept learning and the recognition of common patterns to obtain either complete or partial class templates. This serves as a complement to the Forrest et. al. approach above,. From a heuristic perspective, the De Jong and Spears approach of concept learning results in rules that can be applied to the search in question in order to categorize population members. Further, "supervised concept learning involves inducing concept descriptions for the concepts to be learned from a set of positive and negative examples of target concepts. Examples are represented as points in an n-dimensional feature space, which is defined *a priori* and for which all the legal values of the features are known. Concepts therefore are represented as subsets of points in the given n-dimensional space." [22]

The De Jong et. al. approach expresses the complex concepts involved in adaptive pattern recognition as a "disjunctive set of classification rules[where] the left-hand side of each rule (disjunct) consists of a conjunction of one or more tests involving feature values[and] the right-hand side of a rules indicates the concept (classification) to be assigned to the examples which match its left-hand side. Collectively, a set of such rules can be thought of as representing the (unknown) concepts if the rules correctly classify the elements of the feature space." [22] In the case of spectra, an extremely large number of variables (feature values) could easily increase the complexity of the solution space. Therefore, it is necessary to reduce this number by limiting the features introduced and allowing the GA to further eliminate unnecessary features. "By restricting the complexity of the elements in conjunctions, we are able to use a string representation and standard GAs, with the only negative side effect that more rules may be required to express the concept. This is achieved by restricting each element of a conjunction to be a test of the form:

"return true if the value of feature i of the example is in the given value set; return false, otherwise" [22].

This provides for the construction of classifier rules represented as fixed length strings. As applied to the example classification problem (spectra recognition), these rules can be used to classify spectra based on a diverse set of rules that are obtained via application of a GA.

For example, in the case of spectra to be classified, the left-hand side of rules for a five-feature problem could be represented internally as:

F1	F2	F3	F4	F5
011001101011010	1111111	011	111100	11111

where **F1** represents the location of peaks, **F2** represents intensity of peaks, F3 represents mean amplitude, and so on.

"Notice that a feature set involving all 1's matches any value of a feature and is equivalent to 'dropping' that conjunctive term"; so, in this case, the size of the file is irrelevant to the formulation of the rule. [22] A feature set of all 0's indicates a feature set which does not match (cover) any points in the feature space. "The right-hand side of the rule is simply the class (concept) [or virus class] to which the example belongs. This allows for the complete formulation of rules that take the example form:

```
If (F1 = 011001101011010) and (F3 = 011 or 101) and (F4 = 111100) then
                        acetone detected.
```

In this approach, "each individual in the population is a variable-length string representing an unordered set of fixed-length rules (disjuncts)," allowing for the suitable application of GA operators. [22] It critical to pick a good fitness function which rewards the right kinds of individuals. The fitness F of each individual rule set i is computed by testing the rule set on the current set of examples where

$$\mathbf{F(individual}\ i\mathbf{) = (percent\ correct)}^2.$$

"This provides a non-linear bias toward correctly classifying all the examples while providing differential reward for imperfect rule sets." [22]

2.3 Parallel Computing Concepts

Just as the BIS is distributed in nature, the proposed distributed AIS relies heavily parallel computing concepts. Each node with the AIS plays a role in the overall "health" and performance of the system through independent processing and collective communications. Realization of this model requires the incorporation of an established parallel architecture in order to maintain system control reap the benefits derived from parallel and distributed execution. In the case of the proposed system, two key parallel computing paradigms are observed and implemented: (1) Master-Slave and (2) Island-model communications.

2.3.1 The Master-Slave Paradigm. The master-slave model is necessary to control the flow of warnings and detectors from the global node to network and sensor nodes (and vice-versa). The model itself "is easy to visualize from a management perspective – objective function evaluations (task decomposition) are distributed among several slave [*network and sensor*] processors while a master [*the global node*] executes the evolutionary operators (EVOPs) and other overhead functions" [71]. The model depicted in Figure 10 represents a two-level master-slave configuration; however, the AIS proposed here implements a three-level relationship, where the nodes as level n are considered to "master" nodes at level *(n - 1)* where n is equal to 3.

"Because communication (generally) occurs between the master and slaves at the end of each generation, communication time is most likely not an overriding factor" in the AIS's efficiency as that cost primarily depends on the number of sensor and network nodes, "the hardware architecture, and the communication backbone on

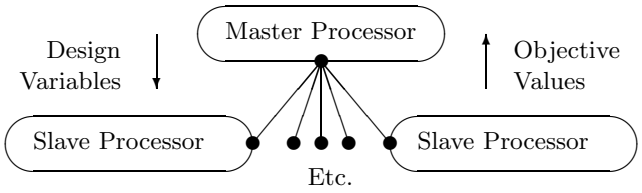

Figure 10 Master-Slave Paradigm [71]

which the algorithm executes" [71]. "As it is generally accepted that real-world objective function evaluations are the most computationally expensive EA components, communication and EVOP cost should then be fairly inconsequential in comparison. This is not meant to imply that scalability is never a problem, but master-slave implementations generally become more efficient as objective evaluations become more computationally expensive. However, note that computational loads must be evenly split among slave processors else significant lag times may exist between generations as the master processor waits for slave evaluations to complete. This 'generation gap' is not wanted! This situation can also occur in a heterogeneous environment" [71].

2.3.2 The Island Model Paradigm. Communications between sensors use the "island" model paradigm, which is "based on the phenomenon of natural populations evolving relatively isolated from each other, such as that occurring within some ocean island chain. [Systems] based on this paradigm are sometimes called 'distributed'...as they are often implemented on distributed memory computers; they are also termed multiple-population or multiple-deme[1]...[systems]" [71]. No matter their name, the key defining characteristic is that individuals within some particular (sub)population (or island) occasionally migrate to another one usually based on some fitness criteria. This paradigm is illustrated in Figure 11.

"Conceptually, within the island model, the overall [sensor set] is divided into a number of independent, separate (sub)populations (or *demes*)" [71]. Each sensor op-

[1]a deme is used here to represent a subpopulation residing on each island

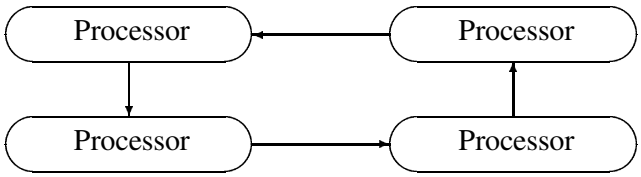

Figure 11 Island Paradigm [71]

erates in isolation for the majority the sensor duty cycle (SDC). Sensors occasionally migrate signatures, measurements, and warning between one particular sensor and its neighbors. The agents sets $A_i = \{D_1, ..., D_n\}$ operate differently within each sensor, thus strongly implying the separate sensors can detect very different regions of the overall solution space [71]. Each sensor could also have "different parameter values as well as a different...structure. This model requires identification of a suitable migration policy to include how often migration occurs..., what number of [detectors] to migrate, how to select emigrating...[detectors] and which detectors are replaced by the immigrants. Relatively thorough gene mixing then exists within individuals in each deme but gene flow is restricted between different demes...Note that an island implementation is sometimes termed *course-grained* parallelism because each island (processor) contains a large number of individual solutions" [71].

2.3.3 The Diffusion Model Paradigm. The diffusion model is used in many parallel computing load balancing applications. The model is based upon the idea of distributing objects (or tasks) to local nodes in the direction of high system loads (or *energy*) to low loads. For example, a system with a high processor utilization rate may wish to disperse its load to a local node if that node has a lower utilization rate in order to improve the overall system performance. System locality is defined by the *neighborhood* set or sets in which the node is a member. For example, Figure 12 demonstrates two overlapping neighborhoods within a mesh of nodes. Node 1 (N1) may choose to diffuse objects to N2 or N3, if node utilization meets a certain threshold to initiate migration [14].

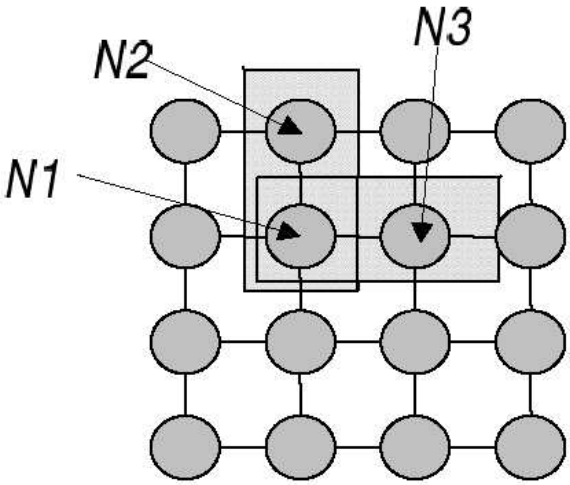

Figure 12 The Diffusion Model Paradigm [14]

While applicable in load-balancing applications, the diffusion model may not be applicable for the purposes of improving AIS classification; and, is not chosen in favor of the island-model paradigm for antibody migration.

2.4 Data Mining / Feature Subset Selection Concepts

Marmelstien [56], the original author of the Genetic Rule and Classifier Construction Environment (GRaCCE) defined data mining as "a broad term used to describe any process that seeks to uncover patterns, associates, changes, anomalies, or statistically significant partitions in data." The traditional method of data analysis is performed manually by developing a hypothesis and then testing if the data supports it. "In contrast, data mining is an automatic process that discovers useful patterns in the data and extracts them" [56]. The data mining process is actually a compilation of six different phases [56]:

Data Selection/Sampling - due to the sheer size of some databases, is often impractical to process them in their entirety; therefore, it is necessary to reduce the data analyzed in some manner or to randomly select a subset of instances for processing.

Cleaning/Preprocessing - the selected data is prepared for processing. This can involve translating the data into an acceptable format or replacing missing or illegitimate entries.

Transformation/Reduction - revise and/or redefine the feature set. All the features included in a given data set are not always required for prediction, and it may often be desirable to create new features to facilitate the mining process.

Data Mining - the application of the selected data mining method to the data.

Evaluation Criteria - the output of the mining algorithm is evaluated against established metrics. This typically reduces the volume of information produced to that which is most relevant.

Visualization - facilitates manual analysis of results by producing information is easily understood.

Figure 13 gives an overview of the data mining process. It should be noted that although the process is depicted serially, the data mining process is actually iterative and repetitive. Many of the phases may be performed in parallel to speed analysis times.

2.4.1 Feature Selection. Selecting of features to be used for accurate classification of a data set can be a difficult task, stricken with both theoretical and computational complexities. "While features within a data set provide the means for discriminating between two classes, too many features can degrade a system's ability to classify data" [68]. This is because the number of samples required for training increases as the number of features and possible values for those features increase, a phenomenon termed the "curse of dimensionality" [68]. For a data set with d features with M possible values for each future, M^d samples are required for training to truly be effective. Further, not all features are required to accurately

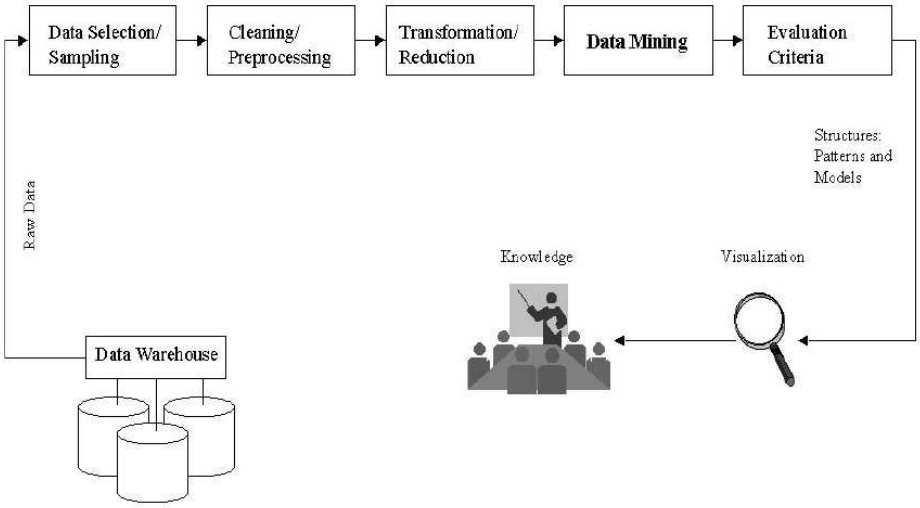

Figure 13 Overview of the Data Mining Process [56]

discriminate between classes. An example of a data set with two classes is shown in Figure 14.

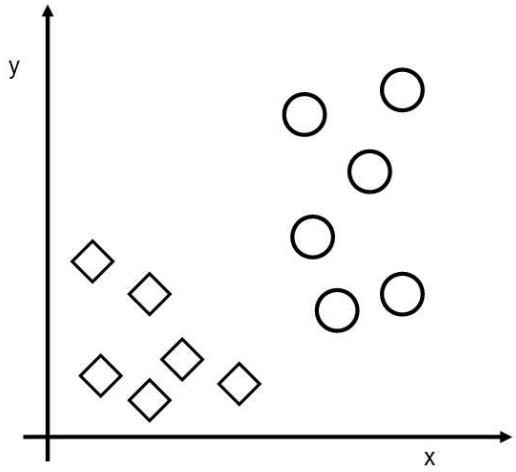

Figure 14 Example of Samples in a Two Classes Data Set

Reduction of the feature set size is one way to fight the curse of dimensionality. Only the x feature in Figure 14 is necessary to discriminate between the triangle and the circle classes. Elimination of the y feature from the data set makes discrimination

relatively trivial after drawing a vertical line through the two data sets, as illustrated in Figure 15.

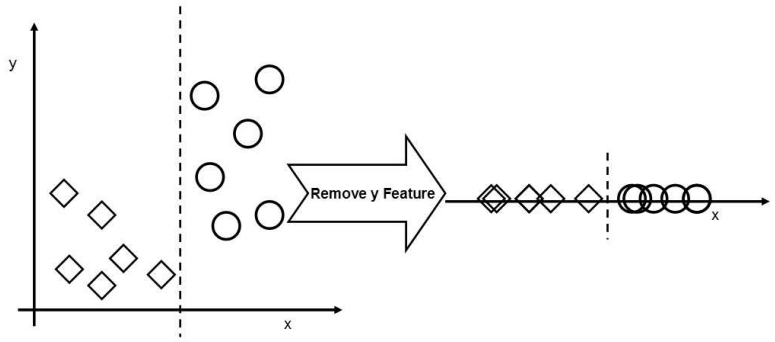

Figure 15 Example of Feature Reduction in a Two Class Data Set

But, given a data set with d features and a desired subset of m features, how is the best subset found? "An exhaustive search would have to try $\begin{pmatrix} d \\ m \end{pmatrix} \frac{d!}{(d-m)!m!}$ possibilities [68]. This results in the need an alternate approach. This research takes the approach pioneered by Marmelstein's Genetic Rule and Classifier Construction Environment (GRaCCE) [56].

2.4.2 Genetic Rule and Classifier Construction Environment. "GRaCCE is a data mining tool that uses evolutionary search techniques to mine classification rules from the data it is given. It is similar to a pattern recognition algorithm, but goes beyond by producing understandable rules used in the recognition" [68]. There are eight different phases of the GRaCCE algorithm, indicated in Table 1 along with their algorithmic complexities. For further information about GRaCCE, see Marmelstein [56], Strong [68], or Yilmaz [74].

2.5 Spectral Analysis

As the purpose of this research is to design a system that recognizes chemicals, the chemical spectra is the chosen method of chemical representation. Spectral analysis, then, is the process by which different measured chemicals (represented by

Phase	**Complexity**
Pre-Processing - Feature Weighting [74]	$O(n^2 + kn)$
Pre-Processing - Feature Selection [68]	$O(n^2 + kn)$
Pre-Processing - Winnowing	$O(dn^2d)$
Partition Generation	$O(2(dn)^2 + nd)$
Data Set Approximation	$O(nlog(n))$
Region Identification	$O(qn^2d^{\frac{3}{2}})$
Region Refinement	$O(dnlog(n))$
Partition Simplification	$O(d^2n^2log(n))$

Table 1 Worst Case Complexity Analysis of GRaCCE [74]

their spectra) are classified. Spectral analysis was introduced by Redner [64] in 1985 and uses the spectrum of the light emerging from a light-field circular polariscope to find the optical retardation of chemical specimens. The retardation, δ, is found by searching for a theoretical spectrum that closely fits the experimental data. If a spectrum is defined by T intensity measurements, then the difference between the theoretical spectrum and the experimental data is defined by Equation 2 [62].

$$D_i = \sum_{n=1}^{T} |I_{theoretical_n} - I_{experimental_n}| \tag{2}$$

The difference, D_i, varies with the substituted value of δ in a complex manner over a wide range of retardations, resulting in very large numbers of calculations where a series of searches are performed in order to find the global minimum. Given this complexity, traditional search methods are ill-equipped to deal with multiple minima that may occur during the search; thus, there is a clear requirement for a fast and efficient iterative algorithm [62].

2.5.1 Existing Methods for Spectra Classification. Four primary methods have been proposed in the past for minimizing the difference found in Equation 2 [62]:

Redner [64] error summation: a large number of theoretical spectra are compared with the experimental data, and those giving the smallest difference are returned as the solution.

Voloshin [72] and Redner database search method: compared experimental data with a database of spectra of a known fringe order.

Sanford and Iyengar [66] method: used the Newton-Raphson method to find a minimum in the difference function.

Haake and Patterson [38] method: used a golden section search to find all of the local minima and then returned the smallest of them as the solution.

2.5.2 Chemical Sensors. Worldwide, sensor research is an extremely hot topic. Sensors are capable of providing humans with data about anything and everything to enhance our ability to control and measure our environment. More related to biological agent classification is the Electronic Nose research being conducted by Ewing, Abdel-Aty-Zohdy, Purdy and a consortium of universities. Other similar research is currently in progress at Caltech, Ohio State, Michigan State, and Wright State University. The goal this research is to devise a system capable of quickly measuring and classifying elements simply by analyzing the resulting chemical spectra plots. Recently, significant progress has been made in this area by incorporating "Systems on a Chip" and nanotechnology, resulting in extremely small but powerful sensors capable of producing highly accurate environmental measurements. Examples of similar sensors developed at Caltech can be seen in Figure 16 and Figure 17.

2.6 Summary

In summary, Chapter 2 has provided a brief background and discussion of a wide range of topics. AIS concepts provide foundation for overall system execution and performance improvements. Parallel computing ties the system together

by providing system control and communications. Data mining concepts reduce the dimensionality of the problem domain, thereby limiting the search space and improving system performance. And last, but not least, sensor research provides system "input", without which the system would could not function. As the focus of this research, the distributed AIS proposed in detailed in next three chapters brings together the complementary characteristics of each of these subjects.

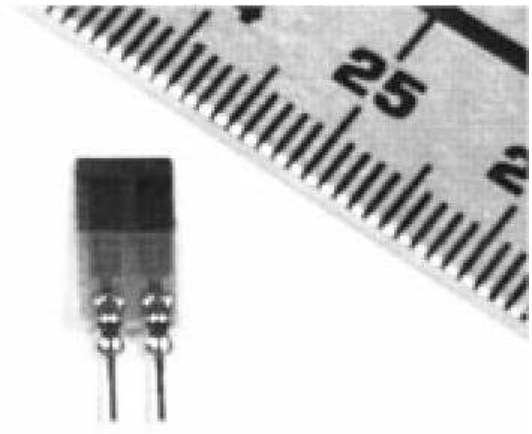

Figure 16 An odor sensor [41]

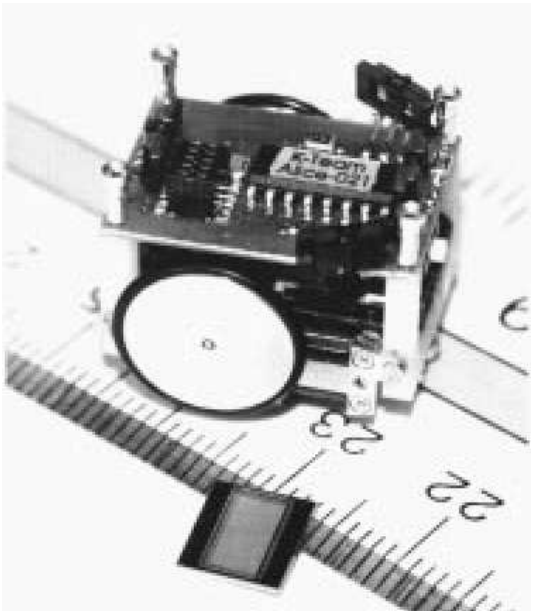

Figure 17 Alice robot with 400 element olfaction chip [41]

III. HIGH LEVEL DESIGN

This chapter provides a high level design for the proposed DAIS. The DAIS problem is decomposed into basic problem sub-domains. Sub-domains are then unified and the operations of the DAIS, as a whole are specified.

3.1 Problem Solution Domain

Full specification of the problem solution requires the definition of three distinct sub-domains: (1) Genetic Algorithms, (2) Artificial Immune System, and (3) Feature Subset Selection. High-level descriptions of each sub-domain are given in the following subsections.

3.1.1 Problem Statement. The DAIS problem concerns the detection and classification of biological agents in a distributed artificial immune system. This problem domain, D_{DAIS}, consists of three separate sub-domains: the AIS sub-domain (D_{AIS}), the genetic algorithm sub-domain (D_{GA}), and the feature subset selection sub-domain (D_{GRaCCE}). In other words,

$$D_{DAIS} = D_{AIS} \cap D_{GA} \cap D_{GRaCCE} \tag{3}$$

Figure 18 illustrates the interactions between signature acquisition, feature subset selection, antibody generation, and DAIS execution. The goal is to continuously improve classification accuracy after each cycle.

It should be noted that although this research presents a strategy for unified system operation and communications between all domains (GRaCCE, Genesis, and the AIS) as shown in Figure 18, these interactions are simulated during system testing (Chapter 5). Testing was conducted over an extended period of time, during which the types of signatures and patterns used were the best available at that moment. For that reason, the data encoding examples given for each sub-domain describe

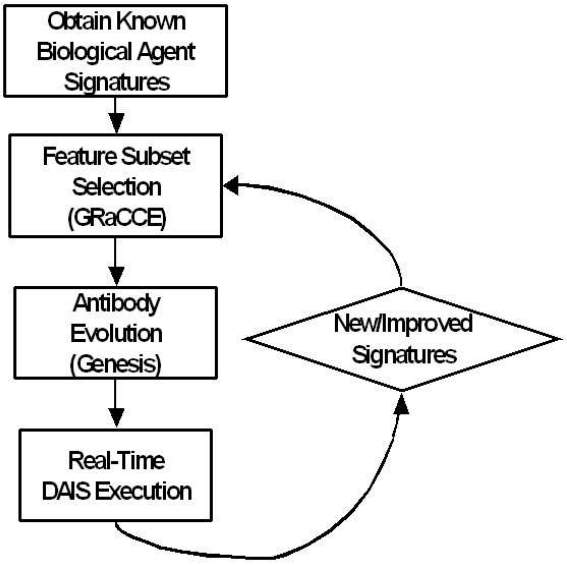

Figure 18 The DAIS Process

different data sets. Though disjoint in regard to using the same data set for all testing, the purpose of validating the strategy for chemical classification presented is still achieved through these pedagogical examples.

3.1.2 D_{AIS} Problem Solution Domain. The AIS sub-domain consists of a defined node architecture that distributes distinct responsibilities to different nodes. In the following subsections, the domain is specified through discussion of the system architecture, node operations, topology, and data encoding. In addition, the range of possible implementation languages and libraries is discussed and choice for each is justified.

3.1.2.1 AIS Architecture. The sensor architecture is based upon the three-tiered model proposed by Lamont, et al [54], and distributes core AIS operations to each layer. The model proposed mirrors an AIS solution for computer virus detection (see Figure 19) that can be easily mapped to the AIS solution proposed which focuses on chemical agent detection.

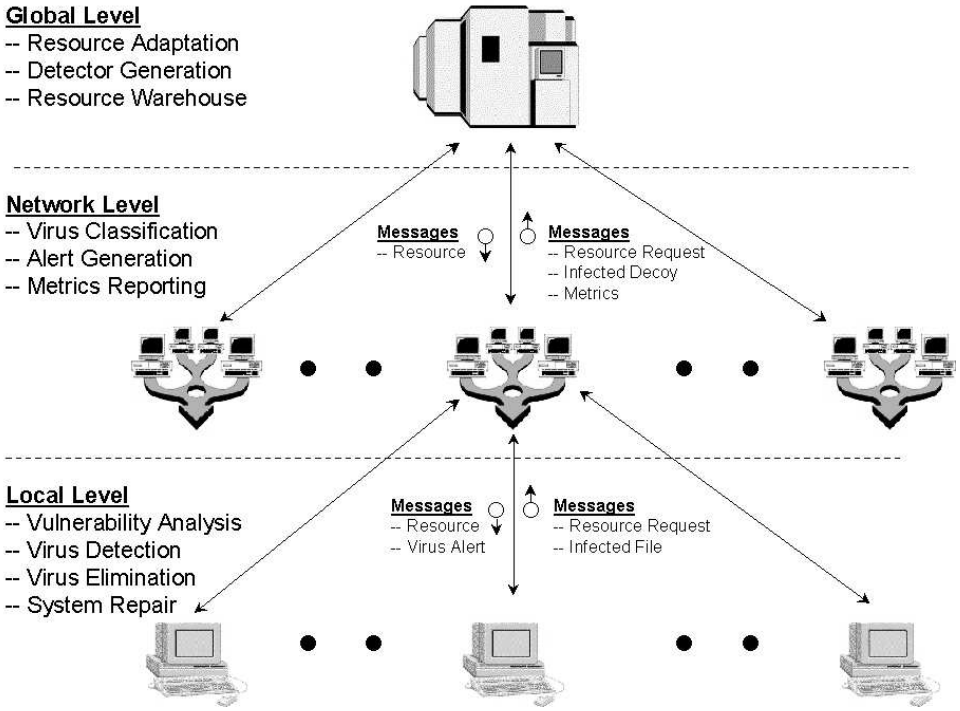

Figure 19 DAIS model sensor architecture [54]

Operating as a Distributed AIS (*DAIS*), the system is comprised of a set of sensors (S), network nodes N), and one or more global nodes (G). Nodes operate in parallel in a hybrid master/slave and island model configuration (see Section 3.1.2.2). As such, the DAIS can be represented as:

$$DAIS = \{S, N, G\}$$

The Local Layer. Each sensor is an individual agent at this level and is singularly responsible for the measurement and classification of elements within the immediate vicinity. This effectively distributes the processing overhead to each sensor, increasing the overall classification power of the DAIS proportionally with $\mid S \mid$. Decisions for *tactical* (local) virus detection are performed at this level. The chief responsibilities of sensors at this level include: taking measurements,

comparing measured spectra to known biological agent plots, and reporting matches (along with the signatures matched) when they occur.

The set of sensors S, is comprised of a set of agents A such that $S = \{A_1, A_2, ..., A_n\}$ where $n = |Agents|$ and agents are comprised of the set of measurements M and detectors D such that $A_i = \{M_i, D_i\}$. One measurement M is taken every Sensor Duty Cycle (SDC) such that $M_i = \{m_1, m_2, ..., m_j\}|j \in SDC$. Further, the set of detectors attributed to agent i is comprised of a set of signatures b to classify measurements during each SDC. This results in the detector set $D_i = \{b_1, b_2, ..., b_z\}$ where z represents the agent memory size.

The Network Layer. The network layer is comprised of more complex nodes represented as the set

$$N = \{N_1, N_2, ...N_b\}$$
$$\forall S \exists N_b \text{such that } S_i \subseteq N_b, \wedge |N| \ll |S|$$

Network nodes help to further classify spectra forwarded to them by sensor nodes. In this capacity, they act as a filter to limit the number of false detections forwarded to the global level. When a valid alert is received from an agent node or a group of agent nodes, the network node forwards this information to other network nodes and to the global node. In addition, network nodes distribute updated detector sets to sensor nodes whenever released by the global node.

The Global Layer. The global layer is comprised of one or more nodes capable of high performance calculations. Each global node is connected to a set of a network nodes such that $G_i = \{N_1, N_2, ..., N_i\}$. This node may be centrally located within a command center; or, may be located remotely when "reach-back" capability has been established. The Global node G is responsible for sensor adaptation, detector (D) generation (Figure 20), costimulation, affinity maturation, and detector memory. The node correlates alerts generated by network nodes and reports

them to command staff.

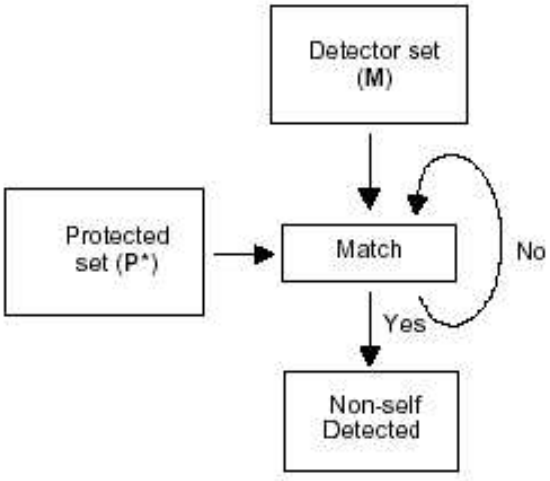

Figure 20 Graphical Depiction of Detector Generation [20]

3.1.2.2 AIS Topology. The physical topology of the AIS radial with connections via wired or wireless means and is depicted in Figure 21. Due to the remoteness of the typical deployed environment, secure wireless communications are preferred to speed system deployment and enhance survivability.

In deference to the physical communications topology, the system takes on an alternate computation paradigm. While all communications flow in a radial/hierarchical fashion as in Figure 21, the logic and control of these messages is quite different. "Three major parallel computational paradigms are considered: 'Master-Slave,' 'Island', and 'Diffusion' paradigms. Others include the 'hierarchical' and/or 'hybrid' paradigms that may be seen as combinations of the three generic forms" [71]. Due to communication and system control requirements, the AIS takes a hybrid master-slave (Figure 10) and island-model (Figure 11) topology. The actual instantiation of this hybrid topology "can be quite simple provided a data structure is utilized that

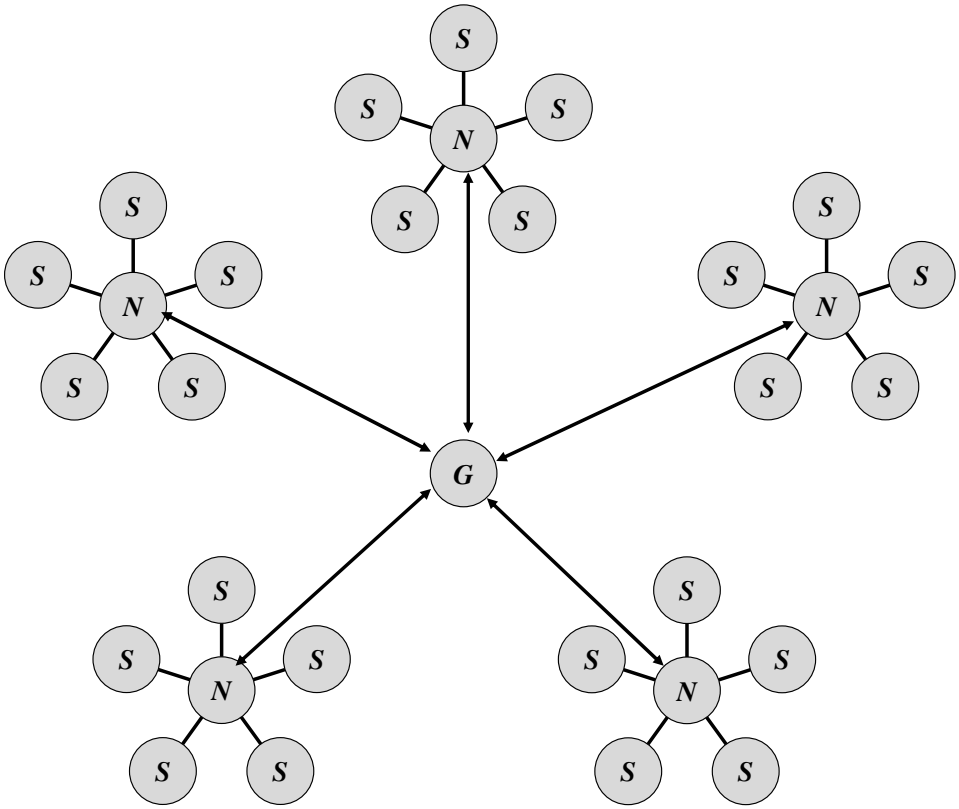

Figure 21 AIS Node Architecture

lends itself to parallelism. Task or data decomposition is another decision that must be made in implementing" a parallel (distributed) AIS [71].

3.1.2.3 AIS Synchronization. Choice of synchronization strategy is also critical to the efficiency and effectiveness of the system. Again, the DAIS takes a hybrid approach, combining both synchronous and asynchronous communications. "Synchronous implementations always deal with same-generation populations with some sort of communication synchronizing all processes. Asynchronous implementations can greatly reduce processor idle time (assuming varying processor speeds/memory/hardware limitations), but this implies that communications occur at random times and sometimes without guaranteed delivery of the messages to the destinations" [71]. All system control communications, such as the distribution of warnings and updated detectors from the global node take place via synchronous

communications under the master-slave paradigm in order to ensure delivery and receipt.

Asynchronous communications take place via the island model paradigm where the only nodes participating in the paradigm are sensor nodes. These asynchronous communications take place whenever a sensor (S_i) detects a biological agent. The measurement(s) (M_i) and detectors (D_i) that exceeded the affinity threshold of S_i are asynchronously passed to adjacent Network Node N_j. This process continues until all sensors are aware of the detection and results in an *immunity response* that raises overall system *affinity* to detect future similar M_i, D_i occurrences. The global node G receives this information after notification from Network nodes and begins production of detectors more capable to produce a system response in the future.

3.1.2.4 Matching Functions. As a core function of the AIS, self/non-self discrimination is made possible through the selection of appropriate rules to determine the degree of similarity or difference between input data and detectors. As the system models self and non-self as series of binary strings, the most appropriate measure of similarity or difference is calculated as the *binary distance* between two specified strings. This measure comes in the following flavors:

Hamming Distance. The Hamming distance measure simply counts the number of bits that differ between two strings. It can be thought of as an "exclusive-or" (XOR) binary operation followed by a binary one summation. This function is signified by equation 4 [59].

$$Hamming\ Distance = \sum_{i=1}^{N} (X_i \oplus Y_i), \qquad X, Y \in \{0,1\}^N \tag{4}$$

Alternate Similarity Measures. Due to the limited ability of the Hamming distance to describe both similarities and differences in one coefficient, other similarity measures have been devised. These measures use the following def-

initions, each adding a different degree of specificity to the resulting measurement [59]:

$$X, Y \in \{0,1\}^N$$

$$a = \sum_{i=1}^{N} \zeta_i, \qquad \zeta_i = \begin{cases} 1: & X_i = Y_i = 1 \\ 0: & otherwise \end{cases}$$

$$b = \sum_{i=1}^{N} \xi_i, \qquad \xi_i = \begin{cases} 1: & X_i = 1, Y_i = 0 \\ 0: & otherwise \end{cases}$$

$$c = \sum_{i=1}^{N} \gamma_i, \qquad \gamma_i = \begin{cases} 1: & X_i = 0, Y_i = 1 \\ 0: & otherwise \end{cases}$$

$$d = \sum_{i=1}^{N} \psi_i, \qquad \psi_i = \begin{cases} 1: & X_i = Y_i = 0 \\ 0: & otherwise \end{cases}$$

The following measures utilize the above definitions of a, b, c, and d to produce better similarity measurements [59, 40]:

$$Russel\ and\ Rao: \ f = \frac{a}{a+b+c+d} \tag{5}$$

$$Jaccard\ and\ Needham: \ f = \frac{a}{a+b+c} \tag{6}$$

$$Kulzinski: \ f = \frac{a}{b+c+1} \tag{7}$$

$$Sokal\ and\ Michener: \ f = \frac{a+d}{a+b+c+d} \tag{8}$$

$$Rogers\ and\ Tanimoto: \ f = \frac{a+d}{a+d+2(b+c)} \tag{9}$$

$$Yule: \ f = \frac{ad-bc}{ad+bc} \tag{10}$$

3.1.2.5 Matching Function Selection. Paul Harmer designed a similar AIS to detect computer viruses in 2000. His research included a thorough comparison study of the above matching measures, as well as others such as *difference matching*,

slope matching, and *physical matching*. For more specifics of this analysis, see the Harmer Thesis [40]; however, his results are summarized in Figure 22. Figure 22 uses a signal to noise ratio (SNR) measurement to determine the specificity or generality of matching functions. "A large SNR indicates a more specific detector, while a low value is indicative of a general detector" [40]. Hence, a detector with a large SNR is able to match non-self with a low false alarm rate, while a detector with a small SNR is more general, and able to cover a larger subset of the self/non-self space [40].

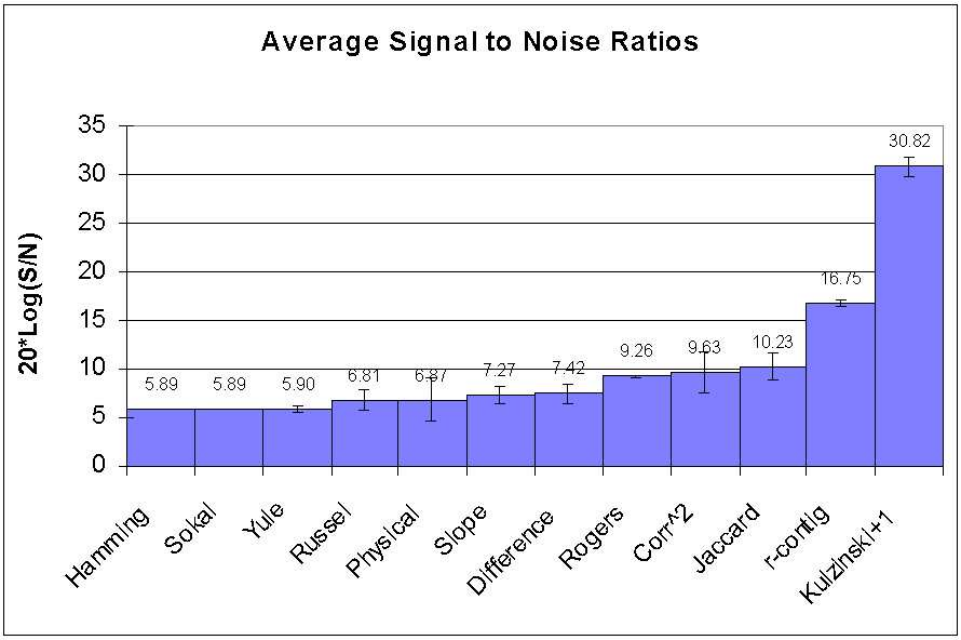

Figure 22 Average signal to noise ratios.

Given the results in Figure 22, the Rogers and Tanimoto matching function is chosen for its ability to provide a good compromise between specificity and generality. This improves the system's ability to cover the space of non-self while limiting the number of detectors necessary to cover this space.

3.1.2.6 AIS Detector Data Encoding. Chemical measurements to be classified must represent real-world chemicals with toxic properties. A list of chemicals considered to be a threat to humans was obtained from the Centers for Disease Control (CDC) [13]. This list was then cross-referenced with a list chemical

signatures available from the National Institute for Science and Technology (NIST) [60], resulting in the following subset of toxic chemicals. Note that methane has been arbitrarily added to increase the overall search space:

- Methane
- Mustard Gas
- Nitric Oxide
- Arsine
- Phosgene
- Hydrogen Cyanide
- Hydrochloric Acid
- Cyanogen Chloride
- Titanium Tetrachloride

The mass spectrum for each of these chemicals contains information on its chemical composition, resulting in a signature that can be used for identification. This plot is the result of the *mass spectroscopy* process. Mass spectroscopy is defined as "a method for experimentally determining isotopic masses and isotopic abundances. A sample of an element is converted into a stream of ions and passed through an electromagnetic field. Ions with different charge-to-mass ratios are deflected by different amounts, and strike different spots on a film plate or other detector. From the position of the spots, the mass of the ions can be determined; from the intensity of the spot, the relative number of ions (the isotopic abundance) can be determined" [46]. Figure 23 is an example of a mass spectrum plot for Mustard Gas. Plots for all chemicals can be found in Appendix A-2.

A large number of features can be derived from this relatively simple chemical signature plot. For the purposes of this AIS, it is beneficial for all signatures to contain information about equal numbers of features. One way to ensure this property

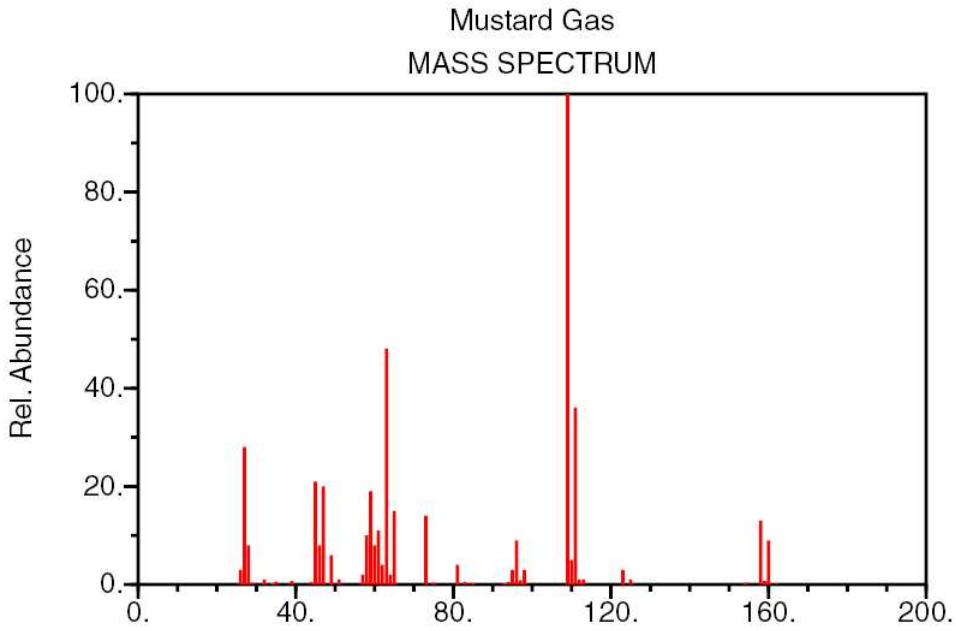

Figure 23 Mustard Gas Mass Spectra Plot [60]

is to derive features from the plot that are common to all other possible mass spectra plots. Such features include the following:

1. First x value
2. Last x value
3. First y value
4. Maximum x value
5. Maximum y value
6. Minimum x value
7. Minimum y value
8. Total Number of Points

All y values in mass spectra plots are a function of relative abundance; that is, the number of ions at that mass number detected relative to the ion detected with the highest abundance. This ion value is always equal to 100% relative abundance.

Therefore, it is not beneficial to use the maximum y value as a feature of the chemical signature. All other features may have some value in discriminating between signatures.

After choosing the appropriate feature subset (subjectively, or by using a data-mining tool such as GRaCCE) , the resulting features are encoded as shown in Figure 24. The resulting signature is simply a binary string representative of the designated features.

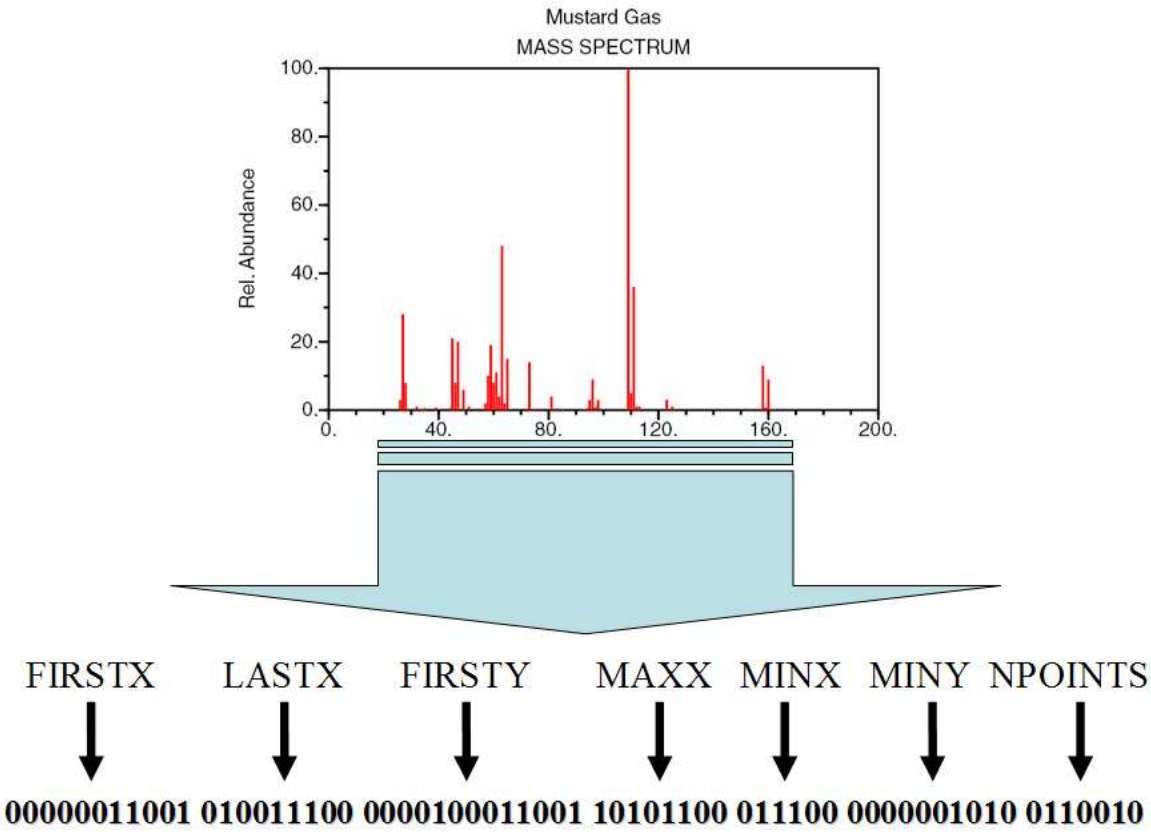

Figure 24 AIS Data Encoding Process

3.1.2.7 D_{AIS} Mathematical Model. A mapping of the general AIS concepts discussed in previous chapters to the chemical spectra classification domain is presented symbolically as follows:

Domains, D

Input D_i

$D_{AIS} = \{S, N, G\}$

S: is a set of sensors

$S = \{A_1, A_2, ..., A_n\}$ where $n = |Agents|$

A: set of Agents $\subseteq S$

$A_i = \{M_i, D_i\}$

M: set of measurements

$M_i = \{m_1, m_2, ..., m_j\} | j \in$ SDC

D: set of detectors

$D_i = \{b_1, b_2, ..., b_z\}$ where $z = |memoryA_i|$ and

N: is a set of network nodes

$N = \{N_1, N_2, ...N_b\}$

$\forall S \exists N_b$such that $S_i \subseteq N_b, \wedge |N| \ll |S|$

G is a set of one or more global nodes

$G_i = \{G_1, G_2, ..., G_i\}$

Output D_o - set of detectors *D'*, and warnings W

Conditions

I(M): M_i measurement, $i \in SDC$

O(W, D'): W warning, D' improved detectors

Operations

Next-State Generator - System Duty Cycle *SDC*

Feasibility(M,D) - $W =$ TRUE iff

$M_o(D, M_i) \geq$ *affinity threshold*, M_o

Solution (D',W): new detectors *D'*, and warnings W

D' generated by D_{GA} and passed

from G to S via N

$W =$ TRUE if $f(D, M) \geq$ *affinity threshold*, $f =$ eq. 9

Objective: $W =$ TRUE when $M_i =$ biological agent

3.1.3 Implementation Languages and Libraries. Realization of the object-oriented DAIS design requires the use of an object-oriented programming language. There are many such languages available. Among the choices are *C++*, Ada 95, and Java. All of these include constructs for object-oriented programming. To enable selection of the appropriate language, a few requirements must be met:

1. **Availability:** Compiler and libraries should be easily accessible and available for download.
2. **Portability:** Code must be able to be executed on multiple system platforms and architectures to enable "heterogeneous" processing.
3. **Programming Environment:** A robust and programmer-friendly programming environment should be readily available to reduce the likelihood of logic errors and speed system development. This environment should be available for free download or included as part of the software on local systems.
4. **Libraries:** A large number of libraries should be freely available to reduce the need for low-level programming constructs in file I/O and communications operations.

Given these requirements, *C++* and Ada 95 are not logical choices. *C++* does include a rather robust programming environment through Microsoft Visual Studio C++; however, this environment uses a compiler that is specific to the Microsoft platform and effectively hinders portability. Ada 95 is no longer in development and available libraries are extremely limited. Further, no up-to-date programming environment exists for Ada 95.

Java is therefore the logical implementation choice for the following reasons:

1. The Java compiler is freely available regularly updated.

2. Java code is highly portable. The same code may be executed on any platform for which a Java compiler is available. Currently this includes all major system platforms.
3. There are a significant number of user-friendly Java programming environments freely available or available through local software distributions. The *IBM VisualAge for Java* programming environment is chosen for its real-time error checking and debugging features.
4. A wide variety of Java libraries can be found on the Internet. The majority of these libraries are already included in the basic Java compiler and the range of common functions.

3.1.3.1 Communications Libraries. As an object-oriented system, the AIS is composed of modules designed around traditional Java object-oriented programming techniques such as inheritance and aggregation to establish the AIS control hierarchy and specify the operations to take place at each node. Establishing communications between nodes requires use of a standardized communications protocol. There are many protocols that meet or exceed the communications requirements of a distributed AIS. However, the chosen communications library must meet a few mandatory requirements [40]:

1. **Efficiency**: Low communications overhead and start up time
2. **1-to-1 Communications**: An ability to send messages directly from one node to another
3. **1-to-Many Communications**: An ability to send a message from one node to many nodes (*multicast*).
4. **Asynchronous Communications**: The ability to send messages without timing constraints

5. **Abstraction**: Messages can be sent without interface to low-level networking commands
6. **Simplicity**: Facilitate robust and reliable communications without complexity
7. **Portability**: Ability to communicate between heterogeneous systems and architectures

Among the many choices of communications libraries compatible with Java, there are a few stand-outs to be considered. These stand-outs include the Java Shared Data Toolkit (JSDT), Java Message Service (JMS) and Java Message Queue (JMQ), and Message Passing Interface (MPI).

JSDT. The Java Shared Data Toolkit supports collaborative applications through an established set of communications constructs [40]. "This set of classes provides an abstraction above the basic networking functionality to offer communication sessions between objects, with each session capable of supporting multiple separate data channels" [40]. The JSDT allows for low-level networking communications that utilize sockets, hypertext transfer protocol (HTTP), lightweight reliable multicast (LRMP), or remote method invocation (RMI) [40]. The protocol also efficiently supports multicast and point-to-point messaging. JSDT objects communicate by subscribing to *channels*. Objects that subscribe to the same channel can communicate via any of the methods described in the previous paragraph. This ability would enable the DAIS communications in that each node would simply subscribe to the channels necessary to setup the hierarchical communications necessary for DAIS operations. Unfortunately, channel subscriptions must be individually instantiated at each node, requiring significant time and effort to set up hierarchical communications. This process could not be automated by any single node in the hierarchy. This limitation would significantly hinder node setup time, slowing down the testing process by requiring user interaction at every node to establish the full communications tree. In order to collect a large number of data

points for statistical testing, this tree must be instantiated at least ten times per test resulting in an inordinate amount of time for data collection.

JMS and JMQ. The Java Message Service and Java Message Queue work together to provide a common framework for interaction among Java applications in an enterprise network [40]. JMS provides a common API framework for message construction, interpretation, and sending. JMQ provides the constructs for message receival. The JMS/JMQ framework consists of message publishers, brokers, and consumers to facilitate communications. For messages to be sent and received, the publisher object submits a message to message broker. This message is then received by a message consumer that has requested a message or shown an interest in messages from the publisher [40]. While providing high degree of abstractness, the JMS/JMQ framework suffers from the same limitations of the JSDT and would require significant operator interaction to establish communications. This limitation also effectively precludes JMS and JMQ from consideration as a the communications library of choice for DAIS implementation.

Message Passing Interface (MPI). MPI is a standard and portable communications library based upon message-passing, and meets the requirements in Section 3.1.3.1 . The library is the result of intense standardization efforts by the MPI Forum to define a portable message-passing system to support parallel applications. Designed to facilitate high performance computing, MPI allows efficient asynchronous and synchronous communications between multiple networked computers.

At start-up, the AIS determines the node object (Global, Network, or Sensor) to be executed on each system based upon the system's *rank* within the distributed architecture. Determining rank requires a native interface to MPI libraries. In Java, this takes place through the Java native interface (JNI) to C and C++ MPI libraries known as *mpiJava* [12] as shown in Figure 25.

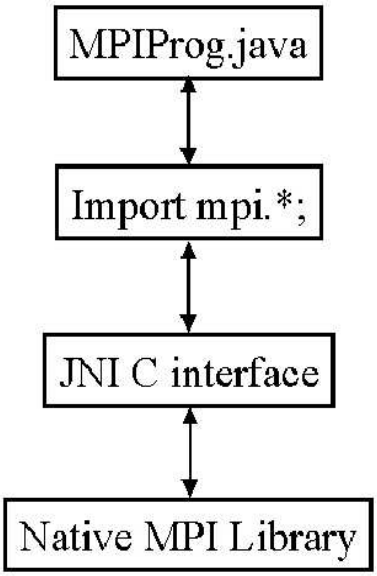

Figure 25 mpiJava implementation layers [40]

3.1.3.2 Size Of D_{AIS} Problem Domain/Search Space. The size of the AIS problem domain and search space is dependent upon the length of self/non-self bit-strings. As such, the search space $O(2^l)$ where l is the length of the longest self/non-self string.

3.1.4 D_{GA} Problem Solution Domain. The GA problem domain includes the operations of a traditional GA (crossover, selection, and mutation) as described in Section 2.1.

3.1.4.1 D_{GA} Mathematical Model. The GA problem domain is defined formally as:

Domains,D

Input D_i

$D_{GA} = (I, \Phi, \Omega, \Psi, s, \iota, \mu, \lambda)$

I: antigen signatures, $(b_i) \subseteq D_i$ (detectors)

Chromosome l composed of attributes

representing *antigens*

$l = B^i$, i length of binary representation

Φ: fitness functions eqs. 12, 13

Ω: GA probabilistic operators

Ψ: generation transition function

s: selection operator

ι: termination criteria

μ: number of parents

λ: number of offspring

Output D_o - set of *antibodies*, $I^\mu = \{b_1, ..., b_\mu\}$

Conditions

I(P(t): population at time t, $t \in SDC$

O(I): improved antibodies

Operations

Next-State Generator - selection $s(P(t), p_s)$,

recombination $r(P(t), p_c)$, mutation $m(P(t), p_m)$

affinity maturation$(P(t))$, negative selection$(P(t))$

costimulation$(P(t))$

Feasibility(I) - $\Phi(I) \geq$ *affinity*

Solution (I): improved *antibody*

Objective: improved $\Phi(P(t))$

3.1.4.2 The General GA Algorithm. A general algorithm for the proposed GA solution is expressed by the pseudocode in Figure 26 [24]. The algorithm is shown graphically in Figure 27.

```
Begin
  t=0;
  initialize P(t);
    where P is the detector population
  initialize A(t);
    where A is the antigen population
  evaluate structures in P(t);
    (According to Φ(P(t), A(t)))
  while termination condition not satisfied do
    begin
      t = t + 1;
      select_repro P(t) from P(t-1);
      recombine and mutate structure
      in P(t); forming P'(t);
      select replace P(t) from P'(t) and P(t-1)
end
```

Figure 26 General GA solution pseudocode

3.1.5 D_{GRaCCE} Problem Solution Domain. The GRaCCE problem domain concerns the selection of n features from an individual with d possible features where $n \ll d$ to reduce the dimensionality of classification algorithm by a factor of $(d-n)$.

Domains,D

Input D_i

$D_{GRaCCE} = (I, F, \Phi, \iota)$

I: antigen signatures, $(b_i) \subseteq D_i$ (detectors)

F: set of features

Chromosome I composed of d features

representing *antigens*

$l = B^i$, i length of binary representation

Φ: fitness function (k nearest neighbor)

ι: termination criteria

Output D_o - set of features F'

such that all I can be classified

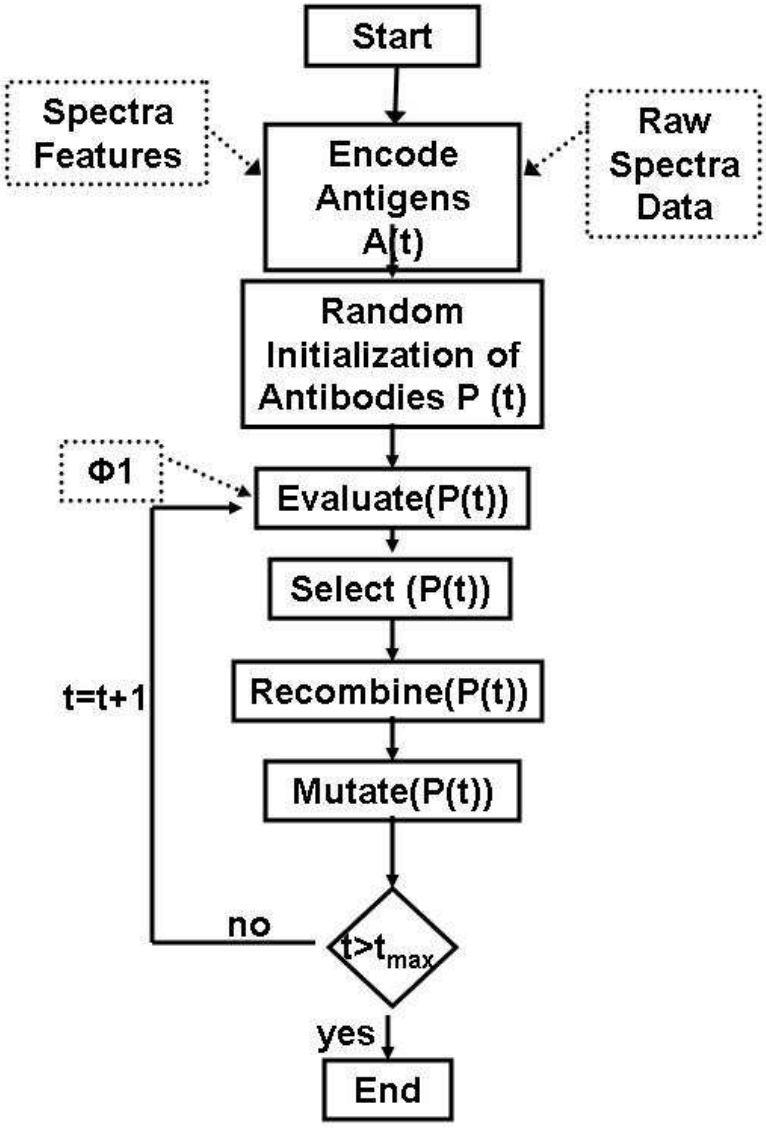

Figure 27 Graphical Depiction of GA Algorithm as Applied to the AIS

Operations

Next-State Generator - feature selection $select(F'$, time $t)$,

Feasibility(F') - $F' \ll F$ and classifies I

Solution (F'): improved F

Objective: improved $\Phi(I)$

3.1.5.1 Simple Vs. Real-World Problem Instantiation. Analysis of the problem from a simple vs real-world perspective reveals a few major differences. First, as tested in a laboratory environment, the system is simple in execution via fully connected high performance computing grid. This grid provides reliable communications on primarily homogeneous systems. Measurements and system "vaccinations" are introduced clinically, as noiseless approximations of the real-world biological signatures. The real-world instantiation is drastically different. In the real-world, communication is both wired and wireless and therefore prone to noise and disruption. Sensors and processing capability are heterogeneous in nature due to the employment of different sensors in different environments and to detect different threats. And finally, the largest difference is in the measurements themselves. These measurements are prone to containing a high degree of noise due to variations in weather, temperature, and air composition.

As a first step towards the realization of this system, the simple laboratory model is a necessary part of the testing process. Experiments attempt to model the real-world instantiation as closely as possible; however, a simulation with real-world accuracy is not possible without the benefit of a real-world system for testing purposes.

3.1.5.2 pGRaCCE Parallelization Concepts. Parallelization of GRaCCE is an interesting topic for the following reasons: (1) the method of execution is highly parallelizable, (2) quick determination of good feature variables enhances an AIS's ability to classify elements and evolve antibodies, and (3) the code has already been previously parallelized by Hammack [39] and therefore easier to characterize.

pGRaCCE Data Decomposition. The serial version of GRaCCE was parallelized by assigning each processor to a class or set of classes (if the number of processors is smaller than the number of classes) to evaluate for classification regions. "The primary reasons for this decision are: (1) low communication re-

quired... [and] (2) a less complex static scheduling scheme could be used" [39]. Data decomposition, then, is accomplished by region assignments to each processor.

pGRaCCE Task Decomposition. Task decomposition is accomplished by assigning the various serial algorithm tasks to each processor so that regions distributed after data decomposition can be assessed in parallel. Figure 28 is a graphical depiction of the serial task implementation.

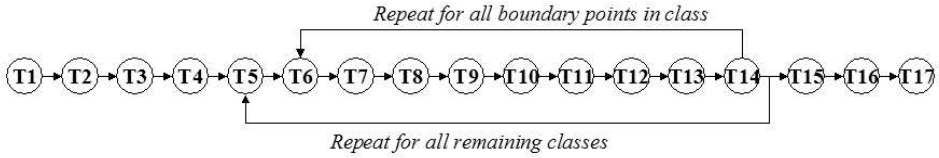

Figure 28 GRaCCE Serial Task Execution [39]

The details of each task depicted in Figure 28 are [39]:

1. GA-based feature selection - selects the best m features
2. Winnowing process - remove all points misclassified by kNN classifier
3. Estimate class boundaries - use estimates to create partitions
4. Compute weight for each boundary point
5. Select target class wt which has not yet been evaluated
6. Choose unassigned boundary point with greatest weight as focus of search
7. Filter out partitions not related to the class of the chosen boundary point
8. Measure partition distance to the boundary point
9. Sort partitions on distance from boundary point
10. Orient partitions such that boundary point, b, has a positive value
11. Find initial solution using a greedy search technique

12. Initialize GA population with results from greedy search
13. Perform GA-based search
14. Assign boundary points within best region found
15. Filter out disproportionately small regions
16. Test and remove extraneous boundaries
17. Recompute the covariance matrix of each region

While all processors perform the same tasks, the tasks are performed on different data sets in parallel. Tasks 6-14 are repeated for the remaining boundary points in target class(es) assigned to each processor. "Once all boundary points...have been evaluated, tasks T5 to T14 are repeated for all remaining classes" [39]. Figure 29 illustrates this decomposition.

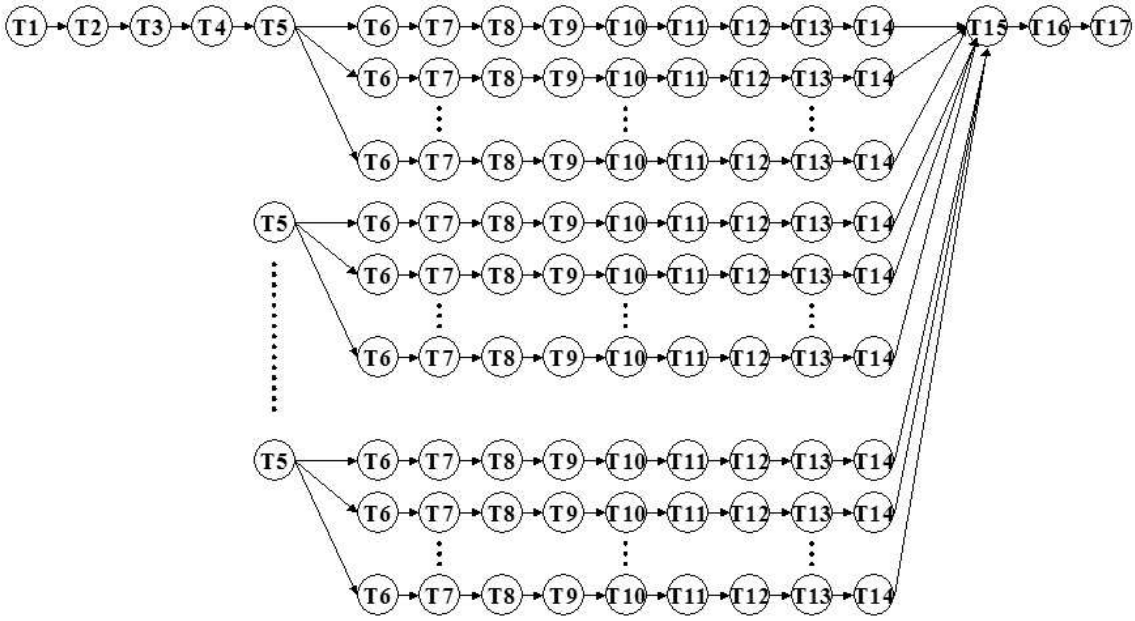

Figure 29 GRaCCE Parallel Task Decomposition [39]

pGRaCCE Load Balancing Approaches. In parallel computing, there are many possible approaches to load balancing. Fortunately, the very low

execution times (1-7 seconds) and small amount of communications associated with pGRaCCE does not necessitate complex load balancing methods. Adding load balancing logic to pGRaCCE would likely only serve to increase the overall runtime, negatively impacting performance.

3.2 D_{DAIS} Problem Solution Domain

The DAIS problem domain is realized through the union of the GA, GRaCCE, and AIS problem domains as stated in Equation 3. Each sub-domain contributes to system performance by reducing the size of the overall search space ($O(2^l)$, l = length of measured bit-strings) and by focusing search in regions that are as close to non-self space as possible. The intersection of these three problem domains is not, however, fluid in nature. GRaCCE must first run independently on a set of antigen signatures in order to determine the minimal feature subsets required for accurate classification. Next, the features not identified by GRaCCE as key to classification must be removed from the original signatures. The resulting signatures are presented to the GA for antibody generation. Finally, evolved antibodies are injected into the AIS for real-time execution. This process is illustrated in Figure 30.

3.2.1 AIS Node Relationships. Relationships between AIS nodes are illustrated in Figure 31. Each node operates independently until receipt of input from adjacent nodes.

3.3 Summary

A high-level design has been presented to describe the basics of DAIS operations. A GA is used to evolve antibodies to classify a given set of antigens while GRaCCE reduces the search space dimensionality. The AIS model is then called upon to provide real-time self/non-self classification with the goal of producing a Warning whenever non-self measurements are encountered. Chapter 4 presents the details of this operation and discusses further implementation issues.

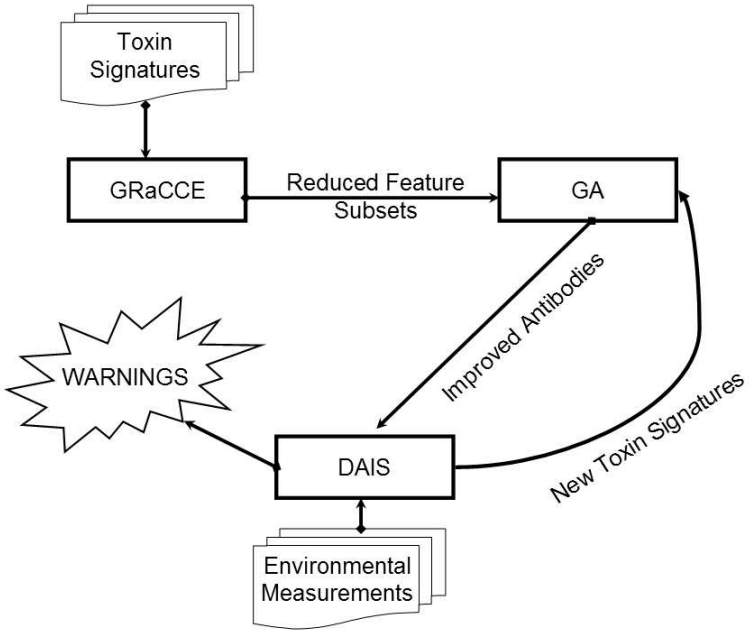

Figure 30 High-Level DAIS Execution

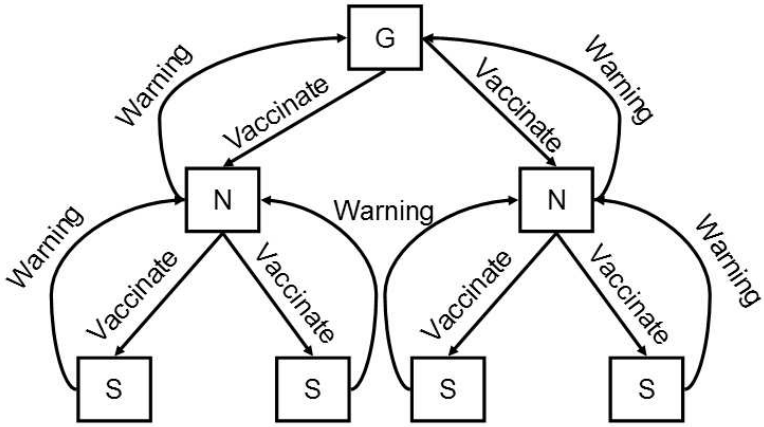

Figure 31 High-Level DAIS Node Interactions

IV. LOW LEVEL DESIGN AND IMPLEMENTATION

Chapter 3 presented the high-level design of the DAIS. This chapter specifies the low-level details for DAIS implementation by addressing the high-level constructs and operations previously presented. Each problem domain is again dissected and key operations are specified symbolically and algorithmically. The data encoding scheme is presented to facilitate a mapping of the problem domain to the algorithmic design. Possible communications libraries are presented and choice of communications library is given. The distributed AIS architecture is made possible through portable source code and standardized communications. Algorithmic details are given for each object within the AIS system design.

4.1 Algorithm Design

The DAIS algorithm design is a low-level mapping of the problem domain to algorithms and data structures. Algorithms for each DAIS sub-domain are presented symbolically followed by their pseudo-code representations. An object-oriented design is presented to allow for abstraction and simplify algorithm specification.

4.1.1 D_{AIS} Algorithm Design. The AIS algorithm requires the generation of signatures (*antibodies*) to classify biological agents (*antigens*). These antibodies are chiefly a result of the negative selection algorithm described in section 2.2.5. The AIS design is the subset of the DAIS design that focuses on the interactions among sensors, network nodes, and the global node. Detection of antigens is associated with an *affinity* threshold statically assigned by the global node at start-up . Antibody affinities are improved within the AIS in a process called *dynamic clonal selection* [50]. Dynamic clonal selection concerns the process of learning normal behaviors by undergoing only a small subset of antigens at one time and replacing antibodies whenever previously observed normal behaviors no longer represent current normal behaviors [50]. Dynamic clonal selection takes place as described by Kim et. al.

[50]. Further, this enhances the effect of *costimulation* and self/nonself determination within the overall system.

At initialization, the AIS is "vaccinated" with antibodies capable of detecting known antigens; a process known as *central tolerisation* [50]. These antibodies are representative of chemical agents known to be used by enemy forces in the area. In a DAIS, the centrally vaccinated nodes distribute improved antibodies to the rest of the system, improving the system's ability to match new antigens. The vaccination of distributed nodes results in a similar affect.

4.1.1.1 D_{AIS} Algorithm Specification. The high-level symbolic formulation of the D_{AIS} specified in subsection 3.1.2.7 is redefined and expanded in the following symbolic specification. Additional details of operations are also specified.

Domains, D

Input D_i

$D_{AIS} = \{S, N, G\}$ where

S: is a set of sensors

M: set of measurements

D: set of detectors

$S = \{A_1, A_2, ..., A_n\}$ where $n = |Agents|$ and

A: set of Agents $\subseteq S$

$A_i = \{M_i, D_i\}$ where

M_i is a measurement associated with agent A_i and

D_i is a detector associated with agent A_i

$M_i = \{m_1, m_2, ..., m_j\} | j \in$ SDC where

m_j is a measurement taken at time $j \in$ SDC

$D_i = \{b_1, b_2, ..., b_z\}$ where $z = |memoryA_i|$ and

b_z is a binary string

N: is a set of network nodes

$N = \{N_1, N_2, ...N_b\}$

$\forall S \exists N_b$such that $S_i \subseteq N_b, \wedge |N| \ll |S|$

G is a set of one or more global nodes

$G_i = \{N_1, N_2, ..., N_i\}$

Output D_o - set of detectors *D'*, and warnings W

Conditions

I(M): M_i measurement, $i \in SDC$

O(W, D'): W warning, D' improved detectors

Objects

GLOBAL-NODE: Highest node in DAIS hierarchy.

Receives warnings from NETWORK-NODES.

NETWORK-NODE: Mid-level node in DAIS hierarchy.

Validates warnings received from SENSOR-NODES

SENSOR-NODE: Low-level node in DAIS hierarchy

Takes measurements and forwards warnings to NETWORK-NODE

Operations

Next-State Generators - System Duty Cycle *SDC*

Generate-Detectors(D): return D'

Correlate-Warnings(N,W): return ALERT

if $\sum W' \geq affinitythreshold$(G)

Monitor-Network-Nodes(N): checks for new W

Monitor-Sensor-Nodes(S): checks for new W

Distribute-New-Detectors(D'): send D' to S

Correlate-Warnings(S,W): return W' to G

if $\sum W' \geq affinitythreshold$(N)

Incorporate-New-Detectors(D): return D'

Take-Measurement: return $M_i, i \in SDC$

Compare-To-Detectors(M,D): return W to N

if $W \geq affinitythreshold$(D)

Feasibility(M,D) - W = TRUE iff

$f(D, M_i) \geq$ *affinity threshold*, f = eq. 9

Solution (D',W): new detectors *D'*, and warnings W

D' generated by D_{GA} and passed from

G to S via N

W = TRUE if $M_o(D, M) \geq$ *affinity threshold*,

M_o = eq. 11

Objective: W = TRUE when M_i = biological agent

4.1.1.2 Object-Oriented Design. Object-oriented design facilitates abstraction and data-hiding to simplify the transition from high-level design, to low-level data constructs, to implementation and coding. From an object-oriented perspective, each node in the AIS executes concurrently while sharing similar detectors generated via negative selection. The system may then be decomposed into three separate modules represented by each type of node: sensor, network, and global. Each node calls upon subsets of the operations defined above. Modules are represented by the following object-oriented pseudocode:

```
OBJECT DAIS(W type Warning,D type Detector)is
  SDC = 0;
    While SDC < MAX-SDC loop
    SDC = SDC + 1;
       GLOBAL-NODE(N);
       NETWORK-NODE(S);
       SENSOR-NODE();
end object DAIS;
```

```
OBJECT GLOBAL-NODE(N Network-Node List) is
  Start-Nodes(Number of Nodes);
  Read-Antigen-File(File);
  Generate-Detectors(D), return D';
  Monitor-NETWORK-NODES(N);
  Correlate-Warnings(N,W); return Alerts
  Distribute-New-Detectors(D');
end object GLOBAL-NODE;

OBJECT NETWORK-NODE(S Sensor-Node List) is
  Read-Antigen-File(File);
  Generate-Detectors(D), return D';
  Monitor-SENSOR-NODES(S);
  Send-Warning(W);
  Distribute-New-Detectors(D);
  Correlate-Warnings(S,W);
end object NETWORK-NODE;

OBJECT SENSOR-NODE() is
  Read-Antigen-File(File);
  Generate-Detectors(D), return D';
  Incorporate-New-Detectors(D); return D'
  Take-Measurement(M);
  Costimulation(W,D);
  Compare-To-Detectors(M,D), return W;
  Send-Warning(W);
end object SENSOR-NODE;
```

Object methods and attributes are inherited from their parent objects. The object relational diagram in Figure 32 illustrates the relationship between each of the objects. Global, Network, and Sensor objects are considered to be Nodes and share all common node operations. Additional object details for these objects can be found in the AIS documentation in Appendix A-3.

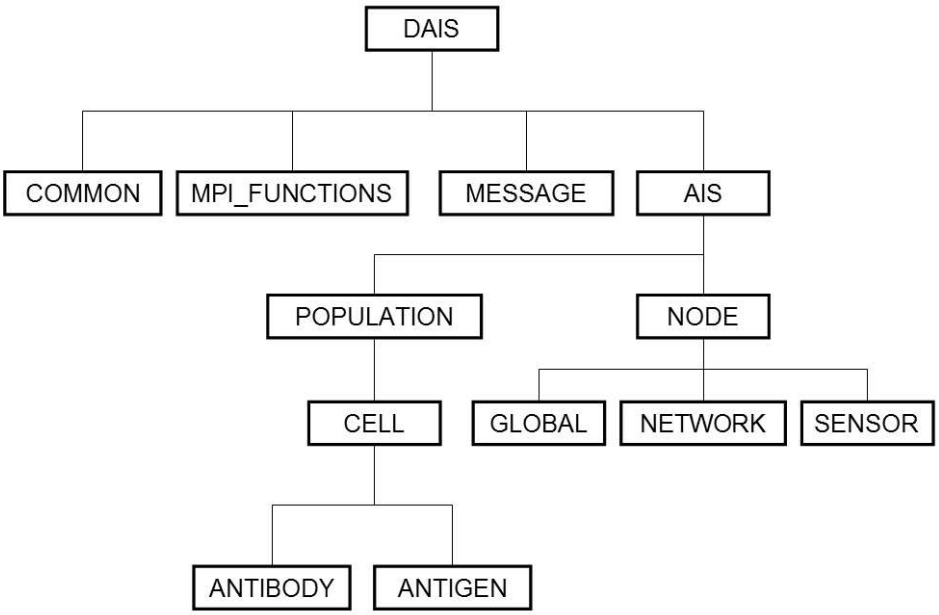

Figure 32 DAIS Object Relational Diagram

4.1.1.3 Object Design Pseudocode. Details of the operations specified for each object in the previous subsection follow.

Start-Nodes(Number of Nodes): Invokes MPI function `MPI_Init(args)` based upon the number of nodes given in command-line arguments. Node objects (Global, Network, or Sensor) are then invoked based upon the rank of each node. A basic algorithm for determination of node object based upon node rank follows:

```
Start-Nodes(NUM_NODES)
  NUM_SENSOR_NODES_PER_NW_NODE = 5;
  NUM_NETWORK_NODES = (NUM_NODES - 1) / NUM_SENSOR_NODES_PER_NW_NODE;
  if (MY_RANK == 0)
    GLOBAL-NODE;
  else if (0 < MY_RANK <= NUM_NETWORK_NODES)
    NETWORK-NODE;
  else
    SENSOR-NODE;
end;
```

Read-Antigen-File(File): Reads antigen file saved as ‘‘`antigens.txt`" and returns POPULATION object containing the given antigens, encoded per Section 3.1.2.6. The antigen file format lists each antigen on a separate line in the following format:

```
Antigen_Name Feature1 Feature2 Feature3 ...  FeatureN
```

Feature values are integers separated by a space up to the total number of features.

Generate-Detectors(D): Returns a specified number of detectors, based upon the `NUMDETECTORS` specified in the ‘‘`config.txt`" file. Detectors are generated via the negative selection algorithm (Section 2.2.5) are bit-strings `ABSIZE` in length.

Monitor-NETWORK-NODES(N): Uses MPI function `Iprobe` to determine whether there is a message waiting. If yes, retrieve the message and process message based upon the type of message received. Messages may be any of the following types: `WARNING, VACCINATE, COSTIMULATE, OR AFFINITY_CHANGE`.

Correlate-Warnings(N,W): Uses *costimulation* to determine whether a warning is legitimate. If the warning received is not detected by a local antibody, the warning is dropped.

Distribute-New-Detectors(D'): Distribute detectors determined to be "good" based upon their affinity to detect antigens.

Send-Warning(W): Sends a warning containing the measurement determined to match a current antibody. Sends message using MPI asynchronous and non-blocking `Isend` command, this allows the node to continue to function without waiting for the message to be received by the destination node. Warnings are always sent one level higher in the node hierarchy; i.e., Sensor nodes only send Warnings to Network nodes and Network nodes only send Warnings to Global nodes.

Incorporate-New-Detectors(D'): Adds D' to current D POPULATION

Take-Measurement(M): Returns a measurement M. M is an antigen `ANTIGENPERCENTAGE`% of the time (specified in ``config.txt"`)

Compare-To-Detectors(M,D): Measurement M is compared to detector population D. If M matches any detector in D with an affinity greater than `MATCHVAL`, a warning is generated. This operation is used in costimulation and negative selection.

4.1.1.4 D_{AIS} Program Variables. AIS execution variables are specified in the `config.txt` file. These variables are:

EXECTIME: The maximum amount of time (in seconds) that the AIS takes measurements, forwarding warnings, and classifying warnings. This time does not include system startup time which is the time it takes to establish all nodes, generate the initial population of antibodies and read the antigen input file.

NUMSELF: The number of "cells" that are randomly generated to represent self. These cells are completely random and are binary strings of length equal to the length of antigen strings.

NUMAB: The number of antibodies that are initially generated via the negative selection process. This number may increase due to affinity maturation and clonal selection.

NUMANTINJECTS: The number of antigens that may be injected into the system. Limits the number of antigen variants.

NUMIMMUNELOOPS: The number of clonal selection loops to perform during each system duty cycle (SDC).

ANTIGENPERCENT: The percentage of measurements taken by Sensors that represent toxic agents.

MATCHTHRESHOLD: The value of the Rogers and Tanimoto (Equation 9) function at which two cells are said to "match".

CLASSIFYTHRESHOLD: The value of the Rogers and Tanimoto (Equation 9) function at which two cells are said to "match", thereby signifying that the cells are in the same class of toxin and classifying the unknown cell.

COSTIMULATIONTHRESHOLD: The value of the Rogers and Tanimoto (Equation 9) function at which two cells are said to "match", thereby costimulating the cells and generating a Warning.

MAXCOSTIMCELLS: The maximum number of antibody cells that may be generated due to clonal selection and affinity maturation.

MAXCOSTIMLIFETIME: The maximum period of time (in seconds) that a non-memory antibody may exist in the system without being costimulated.

ABCELLSTIMULATIONPERCENT: The percentage of antibodies exposed to an antigen during each cycle of the clonal selection loop.

NUMCOSTIMTOWARN: The number of times that a cell must be costimulated in order to generate a Warning.

ABLENGTH: The length of all antibodies. Also dictates the size of the sliding window used to determine a match.

ANTIGENVARIATION: The maximum percent change that may take place in a antigen feature when adding noise to an antigen to produce an antigen clone.

NUMANTIGENMUTATIONS: The number of "noisy" antigen clones to produce per antigen in original antigen file (``antigens.txt"`).

NUMABTOAFFMATURE: The number of antibody clones to produce as a result of affinity maturation.

PROBMUTATION: The likelihood that a given bit is flipped during affinity maturation.

TOTALNUMMEASUREMENTS: The total number of measurement loops that each Sensor executes.

4.1.1.5 D_{AIS} Characteristics and Operators.

- Detector Representation: binary-valued, where $D \in \{0,1\}^l$, $|D| = l$
- Warning Representation: binary vector, consisting of D,
- Fitness: Rogers and Tanimoto fitness function as in Equation 9
- Operations: Costimulation, Negative Selection, Affinity Maturation
- Constraints: D representative of real-world toxic chemical data set (section 3.1.2.6

4.1.2 D_{GA} Algorithm Design. The GA algorithm designs concerns the evolution of detectors (antibodies) to match a given set of antigens. This is accomplished by continually applying GA operations (selection, mutation and crossover)

to a population of detectors for a specified number of generations. The specifics of these operations can be found in Section 2.1.3.

In addition, this process takes place in parallel on multiple nodes. Each node explores a different region of the search space by initializing parallel search with different random seeds. To facilitate this research in parallel GA search, the Genesis [36] GA program has been parallelized by the addition of appropriate MPI function calls to the original source code written by Grefenstette. This results in parallel execution as shown in Figure 33. The instances of the individual algorithm in Figure 33 are copies of the algorithm shown in Figure 27. This results in a broader coverage of the search space which is $O(a^l n)$ where l is the length of detectors and n is the number of antigens.

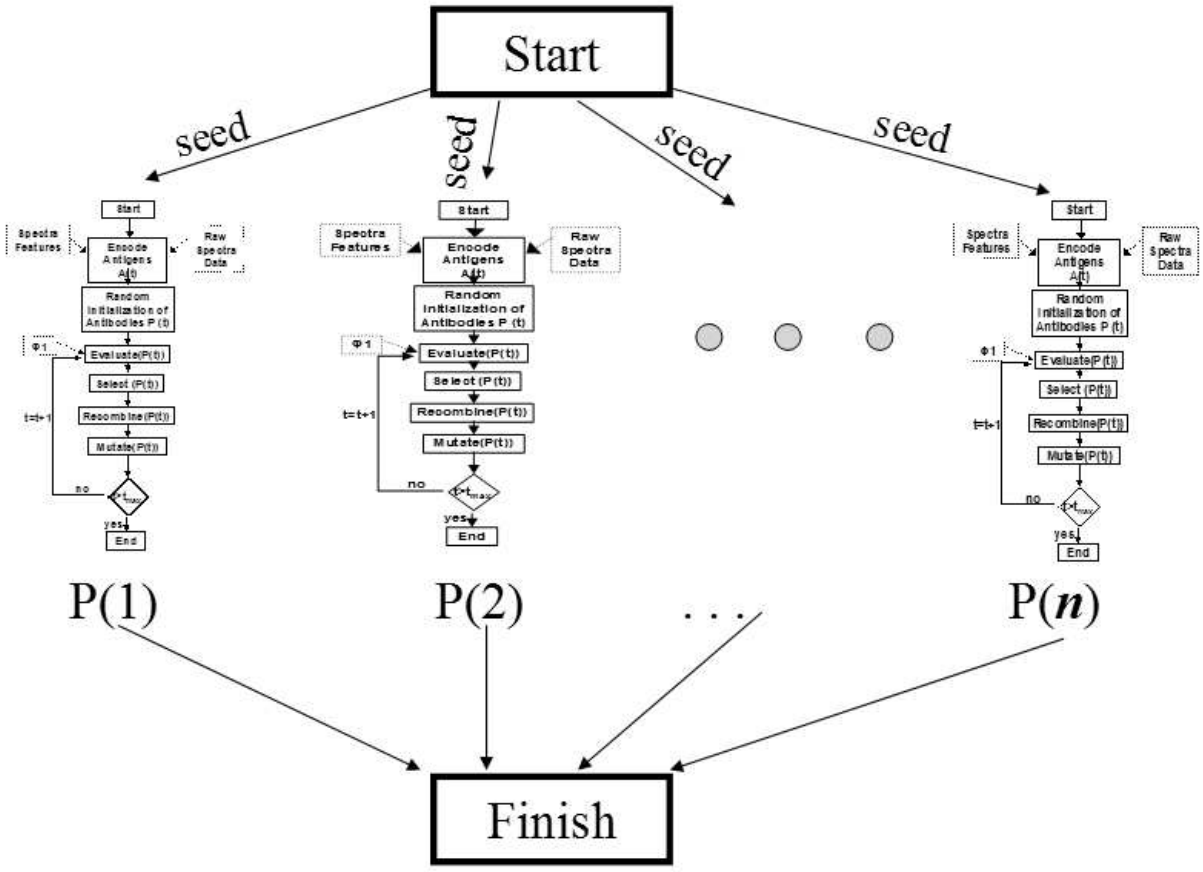

Figure 33 Genesis Parallelization and Execution

4.1.2.1 GA Characteristics And Operators. Even simple GAs are defined by a large number of parameters, operators, constraints, and characteristics. For completeness, characteristics and operators of the Genesis GA algorithm are defined symbolically to standardize their definitions and simplify understanding. GA characteristics and operators are defined as:

- Representation: Binary-valued where $I = B^l$ is an individual chromosome I and B is a binary string of length l.
- Fitness: Scaled objective function value $\forall \vec{b} \in I : \Phi(\vec{b}) = \delta(f(\vec{b}_k(0)), \Theta_\delta)$, where $\delta : \Re \times \Theta_\delta \rightarrow \Re^+$ denotes a scaling function as in $\delta(f(\vec{b}_k(0)), \{c_0, c_1\}) = c_0 \bullet f(\vec{b}_k(0)) + c_1$, where $c_0 \in: \Re - \{0\}, c_1 \in: \Re$ are exogenous constants
- Chromosome: Complete bitstring $\vec{b}$
- Genotype: Partial bitstring $g \subset \vec{b}$ representing a single feature value
- Self-adaptation: Increasing $\Phi(P(t))$ self/nonself discrimination (negative selection)
- Mutation: Bit-inversion, background operator
- Recombination: z-point crossover, uniform crossover, only sexual, main operator
- Selection: Probabilistic, based on elitist strategy: preserves highly-fit individuals
- Constraints: Simple bounds by encoding mechanism for Real to Binary conversion
- Theory: Schema processing theory, global convergence for elitist version

  A detailed formulation of the GA in Bäck's [3] notation follows:

$$D_{GA} = (I, \Phi, \Omega, \Psi, s, l, \mu, \lambda)$$

$$\Leftrightarrow$$

1. $l = B^i$, i length of binary representation,

2. $\forall \vec{b} \in I : \Phi(\vec{b}) = \delta(f(\vec{b}_k(0)), \Theta_\delta)$, where $\delta : \Re \times \Theta_\delta \rightarrow \Re^+$ denotes a scaling function as in $\delta(f(\vec{b}_k(0)), \{c_0, c_1\}) = c_0 \bullet f(\vec{b}_k(0)) + c_1$, where $c_0 \in: \Re - \{0\}, c_1 \in: \Re$ are exogenous constants,

3. $\Phi = \{m_{\{P_m\}}, : I^u \rightarrow I^u, r_{\{P_c,z\}} : I^\mu \rightarrow I^\mu, r_{\{P_c\}} : I^\mu\}$ where the genetic operators are defined as:
$b'_i = \begin{cases} b_i, \ if X > P_m \\ 1 - b, \ if X \leq P_{m_i} \end{cases}$, where $X_i \in [0, 1]$ denotes a uniform random variable sample anew for each string position. The mutation operator $m'_{\{P_m\}} : I \rightarrow I$, uses value swapping for two randomly picked positions within each child that undergoes mutation, producing a string according to $b' = (a'_1, ..., a'_l) = m'_{\{P_m\}}(a_1, ..., a_l) = m'_{\{P_m\}}(\vec{b}) and (\forall i \in \{1, ..., l\})$. The crossover operator $r_{\{P_c,\}}$ denotes a 2-point crossover where $b'_i = \begin{cases} b_{S,i}, \ \forall i(X'_k < i < X'_{k+1}), k \leq 2 \\ otherwise \end{cases}$ and $X'_k < X'_{k+1}$ and $X'_{2+1} = l, k \in \{1, ...l\}$.

4. $\Psi(P) = s(m_{\{P_m\}}(r_{\{P_c,2\}}(P)))$.

5. $s : I^\mu \rightarrow I^\mu$, the proportional selection operator, samples according to the probability density function given by: $p_s(\vec{a}''_k(t)) = \frac{\Phi(\vec{b}''_k(t))}{\sum_{j=1}^{\mu} \Phi(\vec{b}''_k(t))}$

6. The fitness function Φ maximizes the fitness of each I^μ, given N *antigens* detected by $M(i)$ nodes by the ith antibody, we compute the fitness based upon the number of *antigens* it detects, how closely it matches them, and how many other nodes detected this (*antigen*) with this *antibody* [54]:

$$M_o = \sum_i l_i \tag{11}$$

$$\Phi_1 = \frac{1 - SDC}{M_o + 1} \tag{12}$$

$$\Phi_2 = \frac{1}{N}\sum_{j=1}^{M(i)} \begin{pmatrix} 0, & \text{if} M(i) = 0 \\ \frac{1}{M(i)}\sum_{j=1}^{M(i)} \Phi_1 & \text{if} M(i) > 0 \end{pmatrix} \quad (13)$$

7. The termination criterion ι is given by:

$$\iota(P(t)) = \begin{cases} true,\ if\ t > t_{\max} \\ false,\ otherwise \end{cases}$$

4.1.2.2 GA Algorithm. $t = 0$

$initialize : P(0) = \{\vec{b}_1, ..., \vec{b}_\mu\} \in I^\mu$

$I = \{0, 1\}$

$evaluate\{\Phi(\vec{b}_1), ..., \Phi(\vec{b}_\mu\})\}$

$\Phi(\vec{b}_k(0) = \delta(\Phi(\vec{b}_k)), P(0))$

while termination $\iota \neq$ TRUE loop

$for(t = 1 : t_{max})$

$recombine : \vec{a}'_k(t) = r'_{\{P_c,2\}}(P(t) \forall k \in \{1, ..., \mu\} = P'(t)$

$mutate : \vec{a}'_k(t) = m'_{\{P_m\}}(P(t) \forall k \in \{1, ..., \mu\} = P'(t)$

$affinity\ maturation : \vec{a}'_k(t) =$

$m'_{\{P_m + affinity\}}(P(t) \forall k \in \{1, ..., \mu\} = P'(t)$

end

4.1.2.3 GA Data Encoding. As a pedagogical example, acetone and methanol spectra in a noisy environment are chosen as example spectra. These example spectra are shown in Figure 34.

Individual spectra are encoded as 255-bit binary strings. Each five-bit segment represents the binary value of the amplitude (in negative dB) per every 0.1 GHz. For example, the first 0.5 GHz of acetone would be represented as shown in Table 2. Figure 35 shows the full encoded plot of acetone and methanol.

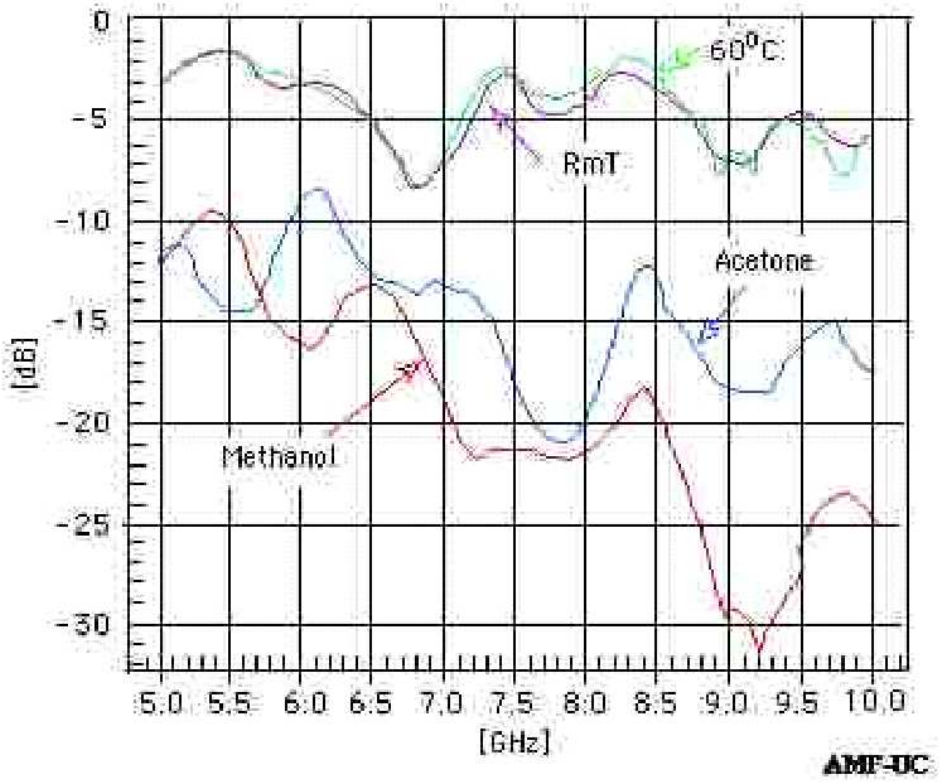

Figure 34 Acetone and Methanol Spectra (Courtesy of the Air Force Research Laboratory)

GHz	5.0	5.1	5.2	5.3	5.4
$-dB^2$	01100	01011	01100	01101	01110

Table 2 Acetone Example Encoding

4.1.3 D_{GRaCCE} Algorithm Design. The GRaCCE algorithm design has been fully specified in the Marmelstein dissertation [56], the Strong masters thesis [68], and Yilmaz masters thesis [74]. Please see these references for further details.

4.2 Summary

This chapter presented the low-level design details for the implementation of the DAIS. Every effort was made to provide the relevant level of detail necessary for full system understanding. Having established the full system design, the next chap-

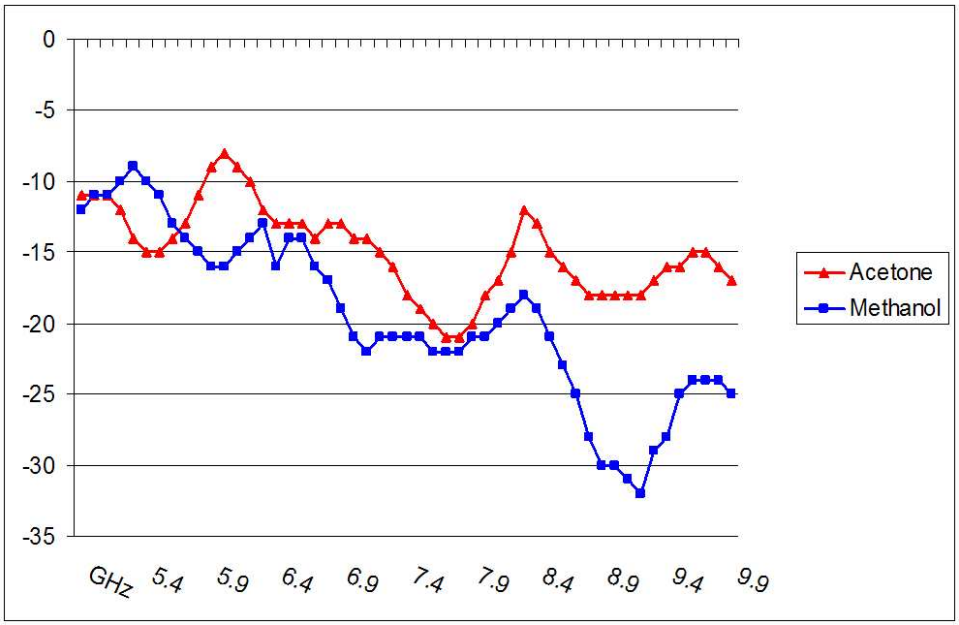

Figure 35 Encoded Plots of Acetone and Methanol

ter provides a design of experiments to fully test system operations and determine the limits of execution feasibility.

V. DESIGN OF EXPERIMENTS

The previous four chapters dissected the DAIS problem domain, mapping it to a high-level design and then the low-level design and implementation details. This chapter discusses the testing of this design by determining the limits of program execution and providing data to enable statistical analysis of performance.

5.1 Performance Metrics

Evaluating the performance of a parallel algorithm such as the DAIS depends not only on the population size, but on the architecture of the parallel computer and the number of processors. Therefore, the DAIS algorithm cannot be evaluated without consideration of the parallel system on which it is executed. This design of experiments explores the performance of each component of the DAIS algorithm on different parallel architectures by first defining appropriate metrics and detailing how these metrics are collected.

5.1.1 Parallel Computing Metrics. To evaluate a parallel system it is necessary to first quantify the performance of a serial implementation of the same system. This provides a baseline for evaluating the gains achieved through parallelization.

5.1.1.1 Speedup. Speedup, defined as the ratio of serial run time, T_s, of the best sequential algorithm to solve a problem to the time to solve the problem on p processors, T_p. Where T_s and T_p are dependent upon the serial and parallel implementation algorithms. For this metric, T_s is the wall execution time for the algorithm on one processor. Therefore, speedup can be calculated as [52]:

$$S = \frac{T_s}{T_p} \tag{14}$$

5.1.1.2 Efficiency. An additional metric for evaluation of parallel algorithm performance is efficiency. Efficiency is defined as a measure of the fraction of time which a processor is fully utilized; or, the ratio of speedup, S, to the number of processors, p [52]:

$$E = \frac{S}{P} \tag{15}$$

Efficiency is another indicator of the scalability of a parallel system. "A scalable parallel system is one in which the efficiency can be kept fixed as the number of processors is increased" [52]. This is calculated via the isoefficiency function, which dictates the growth rate of work required to keep the efficiency fixed as p increases.

5.1.1.3 Startup and Per-Word-Transfer Time. Two additional factors to consider that directly impact the communication time (and therefore the overall parallel processing time T_p) are the startup time t_s and the per-word transfer time, t_w. Startup time is defined as "the time required to handle a message at the sending processor. This includes the time to prepare the message, the time to execute the routing algorithm, and the time to establish an interface between the local processor and the router. This delay is incurred only once for a single message transfer" [52]. Per-word transfer time is defined as the time it takes each word to traverse a link and is computed as $t_w = \frac{1}{r}$, where r is the channel bandwidth in words per second. Because the T_p model used is a 2-D mesh with wraparound, the communications time can be calculated as [52]:

$$2 \cdot t_s(\sqrt{p} - 1) + t_w m(p - 1) \tag{16}$$

Execution time for master-slave communications is easily computed based on the single-objective case [11]. "First, evolutionary operator computation (e.g., selection, crossover, and mutation) time is ignored as it is generally accepted their cost

is much less than fitness computation. Then, given that T_c is communication time between processors, P processors are in use, n is the total population's size, $\sum_{i=1}^{k} T_{f_i}$ is the time required to evaluate one individual for all k fitness functions, and G is the number of generations, the master-slave...running time, T_p^{ms}, may be *estimated* as presented in Equation 17. Such equations can be used to predict the performance over a variety of parallel paradigms" [71]. These parameters are explicitly used in evaluation of parallel GAs.

$$T_p^{ms} = G(PT_c + \frac{n\sum_{i=1}^{k} T_{f_i}}{P}) \; . \tag{17}$$

5.1.2 AIS-Specific Metrics. The effectiveness of the AIS is directly related to its ability to detect anomalous measurements; i.e., measurements that are representative of toxic agents. Quantitatively, system effectiveness is defined as the *detection rate* and *false alarm rate*. These measurements are calculated as shown in Equation 18 and Equation 19, where TP (True Positives) are anomalous elements identified as anomalous; TN (True Negatives) are normal elements identified as normal; FP (False Positives) are normal elements identified as anomalous; and FN (False Negatives) are anomalous elements identified as normal [34].

$$Detection\ Rate = \frac{TP}{TP + FN} \tag{18}$$

$$False\ Alarm\ Rate = \frac{FP}{TN + FP} \tag{19}$$

5.2 Testing Platforms

The GA, AIS, and GRaCCE implementations are tested on systems available in AFIT's High Performance Computing Lab. These systems include clusters of Linux systems connected with Ethernet backplanes. Networked Sun Sparc stations are also used for serial GA testing.

5.2.1 Serial GA Test Platform. The serial version of Genesis is tested on a Sun Sparc10 station with SunOS version 5.8, a 440 MHz UltraSparc 2i processor and 1 GB of random access memory.

5.2.2 AIS, Parallel GA, and GRaCCE Test Platform. The AIS, Genesis parallel version, and GRaCCE are tested on AFIT's High Performance Computing Lab resources "Aspen", and "Poly". These systems are connected via a 100baseT switched Ethernet backbone (Aspen). Table 3 provides additional system details.

Number of Processors	128
Processor	Pentium IV
Clock [MHz]	2000
Cache [KB]	1000
Memory [MB]	2000
I/O Bus	PCI
Local Disk	30 GB IDE
Network	Myrinet

Table 3 Parallel Testing Platform Specification

5.3 D_{GA} Design of Experiments

The GA domain design of experiments tests the performance of the serial and parallel Genesis implementations.

5.3.1 Serial Design. In order to thoroughly test the parallel implementation of Genesis, it is necessary to first determine the best values for algorithm variables such as the probabilities of mutation, selection, and crossover. Serial performance using variables derived from initial testing is then tested on different sets of antigens to quantify performance.

Serial Genesis design of experiments includes a limited benchmark evaluation, in order to determine the best values for the probability of mutation and for the probability of selection. All tests were executed on Sun Sparc stations with SunOS version 5.8 using the Genesis genetic algorithm software[13]. Genesis was chosen due

to its ease of use and the ability to customize the evaluation function. The program produces detailed reports that include mean and average performance and variation for each generation. Details of benchmark tests are shown in the Tables 4 through 7. These parameter values are chosen based upon previous experiences with similar GA algorithms and the results of similar GA experiments in literature [70] [69] [61].

Variable	Tests 1 to 3 (Benchmarks)
T(max)	10000
$\|Antigens\|$	1
$\|P\|$	15
Length Antibody	255
Length Antigens	255
Prob. Mutation	Test 1: 0.0005 Test 2: 0.001 Test 3: 0.0005
Crossover Rate	Test 1: 0.6 Test 2: 0.6 Test 3: 0.8
Replacement	Steady State
Number of Experiments	10
Antigen Type	Benchmark: -11111...00000

Table 4 GA Serial Benchmark Tests

Next, having obtained parameter values that returned the highest fitness values in experiments 1-3, benchmarks for acetone and methanol are obtained by running the GA in search antibodies using these values. Details of these test are shown in Table 5 and Table 6.

Finally, a test was conducted in the presence of both acetone and methanol antigens. This evolved a population of "generalist" antibodies to detect both elements. Details are shown in Table 7.

Variable	Test 4 (Acetone)
T(max)	10000
$\|Antigens\|$	1
$\|P\|$	15
Length Antibody	255
Length Antigens	255
Prob. Mutation	0.001
Crossover Rate	0.6
Replacement	Steady State
Number of Experiments	10
Antigen Type	Benchmark: -Acetone

Table 5 Acetone Benchmark Test

Variable	Test 5 (Methanol)
T(max)	10000
$\|Antigens\|$	1
$\|P\|$	15
Length Antibody	255
Length Antigens	255
Prob. Mutation	0.001
Crossover Rate	0.6
Replacement	Steady State
Number of Experiments	10
Antigen Type	Benchmark: -Methanol

Table 6 Methanol Benchmark Test

Variable	***Test 6 (Acetone and Methanol)***
T(max)	10000
$\|Antigens\|$	1
$\|P\|$	15
Length Antibody	255
Length Antigens	255
Prob. Mutation	0.001
Crossover Rate	0.6
Replacement	Steady State
Number of Experiments	10
Antigen Type	Benchmark: -Acetone and Methanol

Table 7 Acetone and Methanol Benchmark Tests

5.3.2 Parallel GA Design of Experiments. The parallel implementation of Genesis was then tested. Based upon the determination of good variable values in Tests 1 through 3 of the serial implementation, a simple test was conducted to test the algorithm's ability to evolve antibodies capable of detecting antigens that have 1's in their first half and 0's in their second half (or 1111...000). This produced antibodies that resulted in a string of 1's when XOR'd with the antigen. Experiment details are listed in Table 8.

Variable	***AIS Test***
T(max)	10000
$\|Antigens\|$	1
$\|P\|$	15
Length Antibody	255
Length Antigens	255
Prob. Mutation	0.005
Crossover Rate	0.6
Replacement	Steady State
Number of Experiments	10
System	Aspen (Redhat Linux)
Num Processors	2, 4, 8, 10, 12, 14, 16
Antigen Type	Benchmark: 1111...0000

Table 8 Parallel GA Design of Experiments

5.4 D_{AIS} Design of Experiments

The AIS DOE seeks to maximize the Detection Rate (eq. 18) while minimizing the False Alarm Rate (eq. 19). In order to accomplish this goal, it is necessary to determine the best values for system variables described in Section 4.1.1.4. These values are determined by assessing their impact on the DR and FAR. Due to their direct impact on classification, the variables most likely to influence efficiency are: (1) ABLENGTH, (2) MATCHTHRESHOLD, (3) COSTIMULATIONTHRESHOLD, (4) NUMAB, (5) NUMSELF, (6) NUMANTINJECTS, and (7) TOTALNUMMEASUREMENTS.

ABLENGTH: Determination of a good antibody length

Variable:	***1***	***2***	***3***	***4***	***5***	***6***	***7***
Test 1	16, 32, 48, 64	0.5	0.9	10	500	5	1000

MATCHTHRESHOLD: Determination of a good match threshold

Variable:	***1***	***2***	***3***	***4***	***5***	***6***	***7***
Test 2	T(1)	0.9, 0.8, 0.7, 0.65, 0.63	0.7	10	500	5	1000

COSTIMULATIONTHRESHOLD: Determination of a good costimulation threshold

Variable:	***1***	***2***	***3***	***4***	***5***	***6***	***7***
Test 3	T(1)	T(2)	0.9, 0.85, 0.8, 0.7	10	500	5	1000

NUMAB: Determine how the number of antibodies affects Detection Rate

Variable:	***1***	***2***	***3***	***4***	***5***	***6***	***7***
Test 4	T(1)	T(2)	T(3)	10,25,50,100	500	5	1000

NUMSELF: Determine the impact the size of self has on Detection Rate

Variable:	***1***	***2***	***3***	***4***	***5***	***6***	***7***
Test 5	T(1)	T(2)	T(3)	T(4)	500,1000,2000,4000	5	1000

NUMANTINJECTS: Demonstrate effect of a more diverse set of antigen measurements on Detection Rate

Variable:	***1***	***2***	***3***	***4***	***5***	***6***	***7***
Test 6	T(1)	T(2)	T(3)	T(4)	T(5)	1,5,10,20	1000

TOTALNUMMEASUREMENTS: Determine impact of the total number of measurements taken on performance

Variable:	***1***	***2***	***3***	***4***	***5***	***6***	***7***
Test 7	T(1)	T(2)	T(3)	T(4)	T(5)	T(6)	500,1000,2000,5000

Variables: (1) ABLENGTH, (2) MATCHTHRESHOLD, (3) COSTIMULATIONTHRESHOLD, (4) NUMAB, (5) NUMSELF, (6) NUMANTINJECTS, (7) TOTALNUMMEASUREMENTS

Table 9 AIS Design of Experiments

Table 9 indicates the values of each of these variables for a series of tests conducted to quantify the impact of each variable on system effectiveness variables in this table are numbered as in the previous sentence. Each test was run 10 times, resulting in average values for the Detection Rate and False Alarm Rate as well as variance.

5.5 D_{GRaCCE} Design of Experiments

Hammack's [39] parallelized implementation of GRaCCE (pGRaCCE) was evaluated using AFIT's high performance computing lab resources (Section 5.2.2. The "th513" data set available at the University of California Irvine is used. This data set has 5 different possible classes. Due to GRaCCE's task decomposition strategy of distributing class evaluations to separate nodes, the maximum number of nodes that can be tested is 5. This test is also conducted on the "Poly" system, in addition to Aspen. The Poly system is nearly identical to Aspen; but has AMD Athlon processors instead of Pentium IV processors and has a 100BaseT Ethernet backplane. Due to low execution times it is possible to obtain statistically significant results by conducting 30 iterations of each experiment in an inclusive range of 1 to 5 nodes. Table 10 lists details of the experiment.

System	**Num Nodes**	**Num Experiments**	**Num Epochs (GA)**	**data set**
Aspen	1, 2, 3, 4, 5	30	10, 100, 1000	th513
Poly	1, 2, 3, 4, 5	30	10, 100, 1000	th513

Table 10 GRaCCE Design of Experiments Details

5.6 Summary

This chapter described the metrics and testing process for quantifying the effectiveness of the DAIS. A design of experiments was presented for evaluation of GA, AIS, and GRaCCE implementations. Each implementation was tested to obtain data necessary to compute metrics that can be used for thorough analysis of

performance. The next two chapters present the results and analysis of these tests followed by conclusions and recommendations.

VI. RESULTS AND ANALYSIS

This chapter presents the results of the experiments specified in the previous chapter. These results are then analyzed to derive quantifiable characteristics that describe the performance of each system implementation. Results indicate that all implementations perform as designed and return data that is promising for inclusion in DAIS operations.

6.1 GA Results & Analysis

Results of GA testing are shown in Figures 36 through 46. Due to a high degree of variance in Figures 36 through 43, the variance value at each generation is indicated by the lowest line on the graph in order to aid in visualization. Variance in Figures 44 through 46 was small and is indicated by error bars.

Tests 1 through 3 focused on determining good values for algorithm variables such as the probability of mutation and crossover. Based upon these tests, it was determined that good values were 0.001 for probability of mutation and 0.6 for probability of crossover. Graphs representing the results of Tests 1 through 3 are shown. Test 2 produced the highest average match score of 234.10 after 10 executions of 10,000 generations. Test 2 also maintained the lowest overall variance while tests 1 and 3 showed a high degree of variance. While high variance indicates a broader search of the domain space, it also produced worse results by destroying good population members through high crossover (Test 3).

Tests 4 and 5 established benchmarks for the evolution of antibodies to detect Acetone (Test 4) and Methanol (Test 5) in isolation. Test 4 resulted in a match value of 230.60 after 10 executions of the GA, while test 5 resulted in an average match value of 231.50 after 10 executions.

Test 6 demonstrated the ability of the GA to evolve antibodies to detect the presence of both Acetone and Methanol. The results of this test can be seen in Fig-

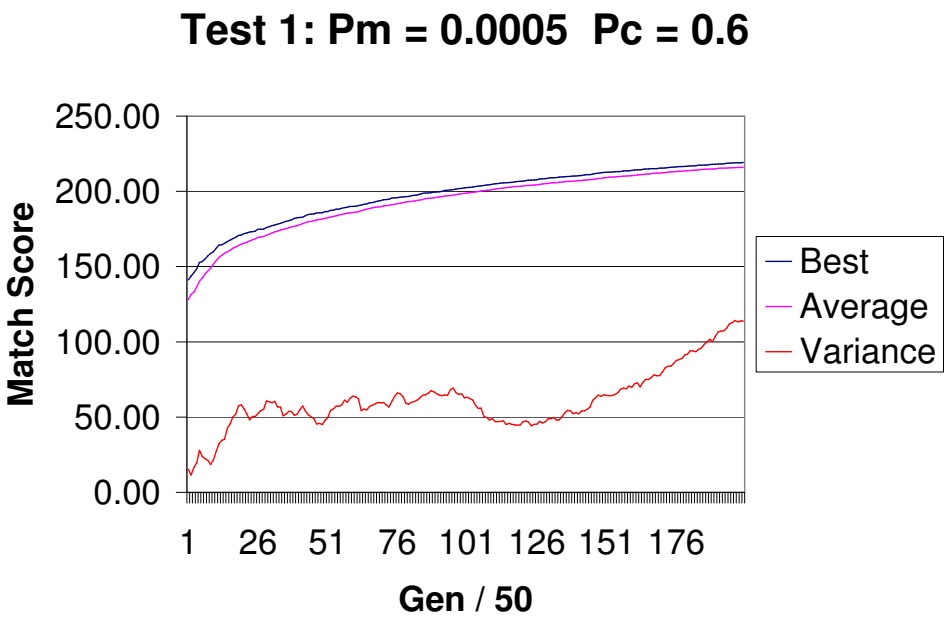

Figure 36 Test 1 Benchmark

ure 41. The test produced antibodies that detected both elements with an average match score of 188.80 over 10 executions. This result was lower than the benchmarks obtained from Tests 4 and 5, because the evolved antibodies must match both elements. Variance decreased as the population zeroed in on a good solution and the slope of the curve appears to still be increasing at 10,000 generations. It is highly probable that a better match score may have been reached by increasing the number of generations in each execution. This was validated by a run to 30,000 generations that actually produced a string with a fitness of 194.5.

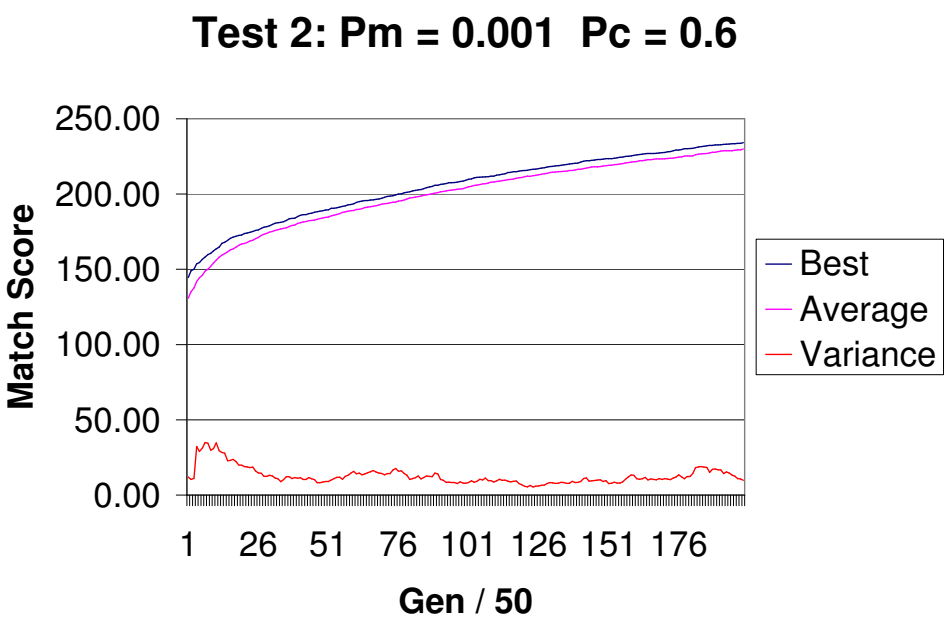

Figure 37 Test 2 Benchmark

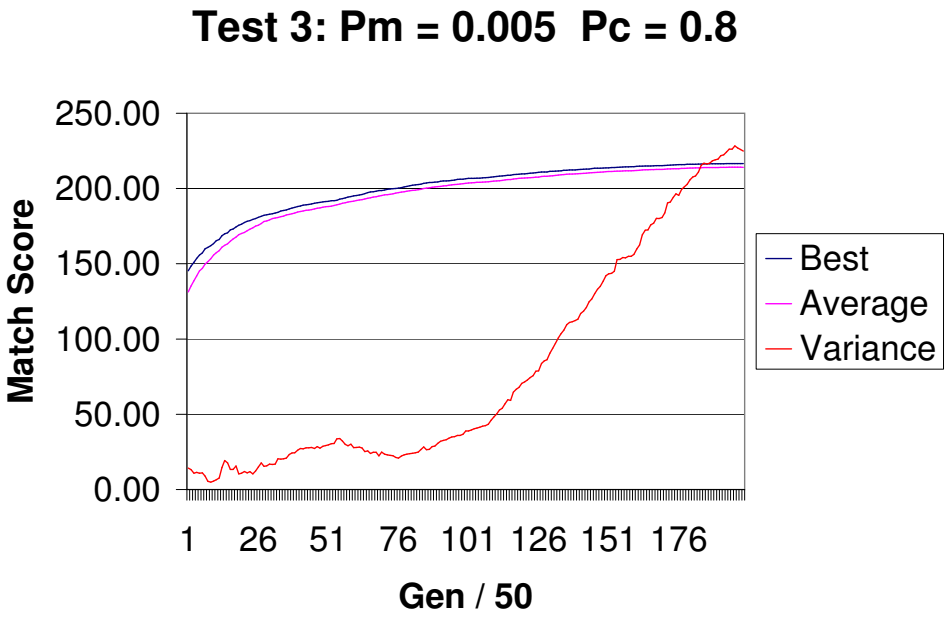

Figure 38 Test 3 Benchmark

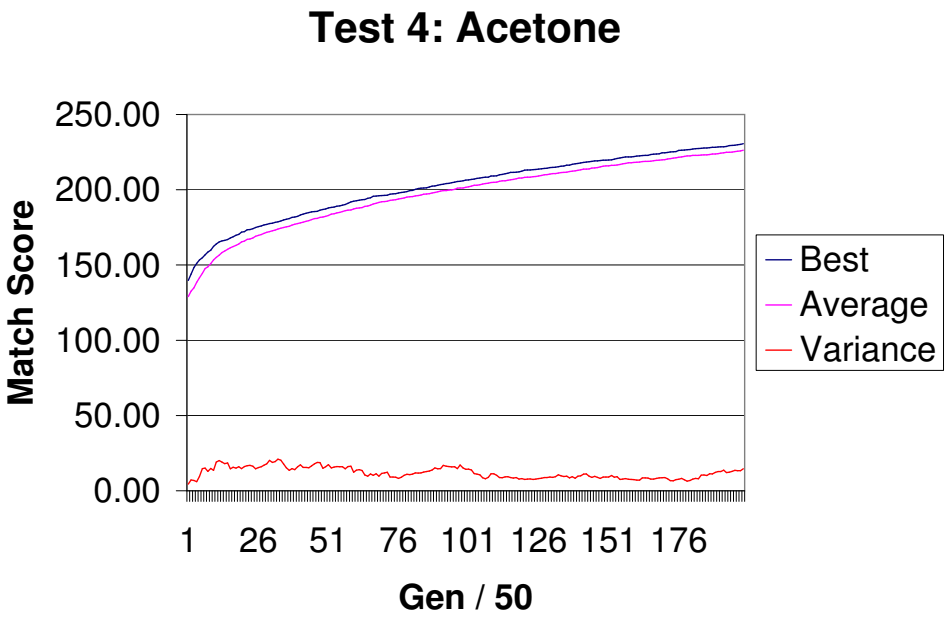

Figure 39 Test 4: Acetone

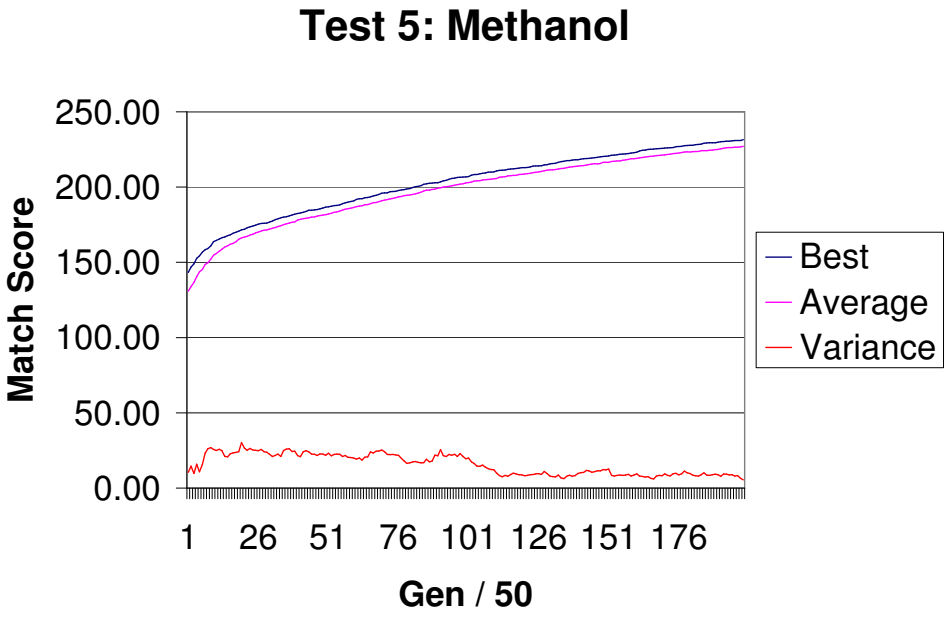

Figure 40 Test 5: Methanol

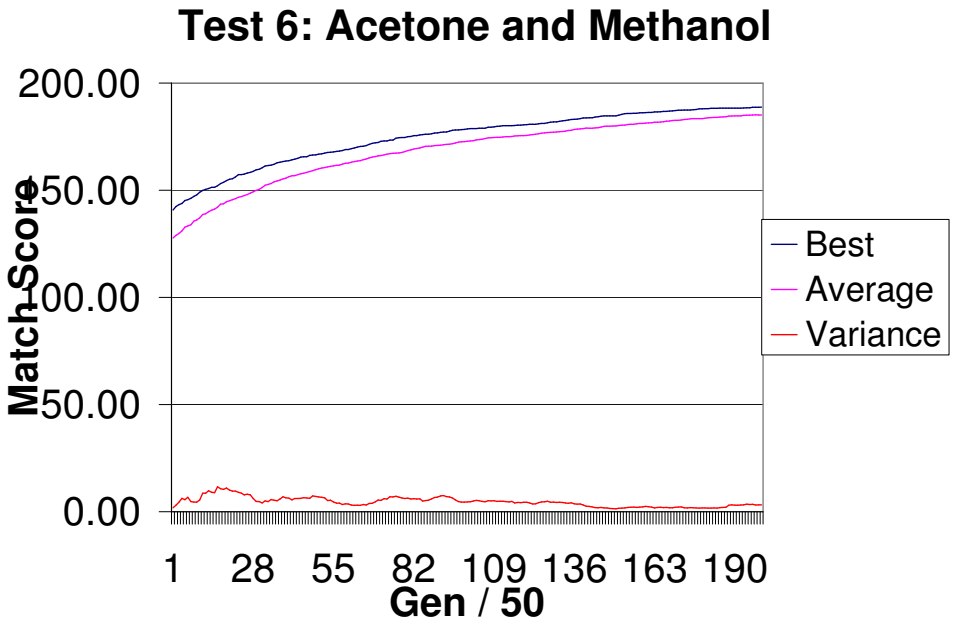

Figure 41 Test 6 Acetone and Methanol

6.1.1 D_{GA} Performance Metrics. Metrics calculated for each of the GA experiments include mean fitness, best fitness, variance, speedup, efficiency, and effectiveness. Details concerning how these metrics were calculated can be found in section 5.1.1

6.1.2 Parallel GA Results & Analysis. As expected, the GA was able to discover antibodies capable of detecting the test antigen. Results of experiments demonstrated the ability of a GA to return a population of antibodies capable of aptly detecting the desired antigens.

Figures 42 and 43 demonstrate the fitness improvement of antibodies after each generation. Note that there is no marked difference between the 2 processor fitness plot and the 16 processor fitness plot. Even though every processor started their initial populations with a different random number seed, resulting in exploration of different regions of the search space, this is likely due using the same values for

crossover, selection and mutation on each processor. Also note that the standard deviation (the bottom dotted line) is very small throughout program execution.

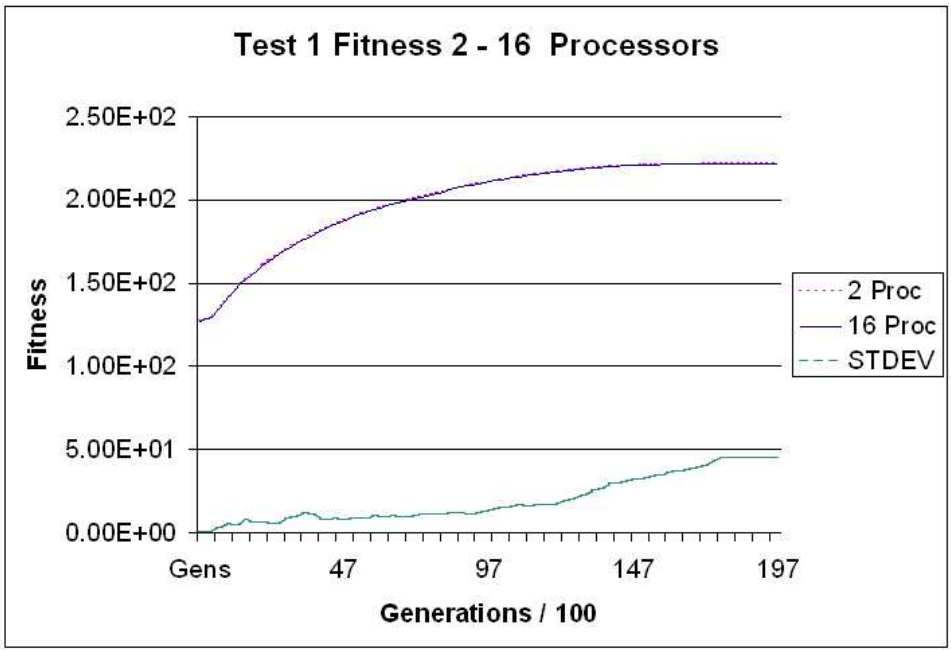

Figure 42 Parallel GA Fitness for 2-16 Processors

Figure 44 demonstrates the speedup of the parallel GA. Note that fitness is steadily decreasing until it reaches 16 processors. This is most likely due to the increasing startup time required as the number of processors increases. However, note that all values are relatively small, given the that a linear speedup would have produced a value of 16 for 16 processors, and this implementation produces a value close to 1.0.

Efficiency (Figure 45) gradually declines as the number of processors is increased; again, most likely due to increased startup and termination overhead.

High system effectiveness is goal of any system. In this case, effectiveness is equal the ratio of max fitness obtained by n processors to the max fitness possible (in this case, 255). Figure 46 demonstrates that increasing the number of proces-

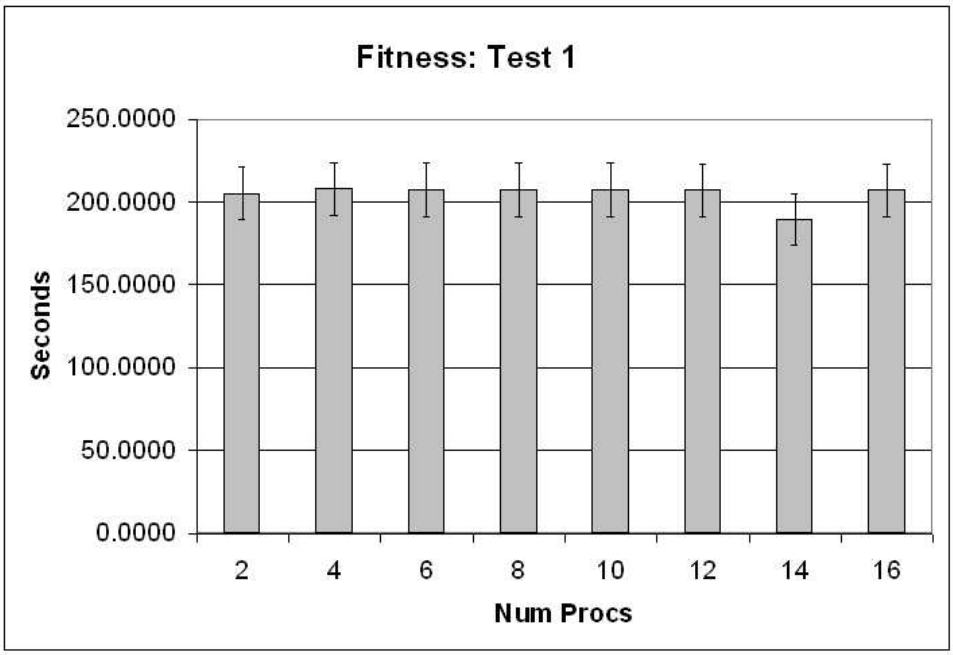

Figure 43 Maximum Parallel GA Fitness for 2-16 Processors

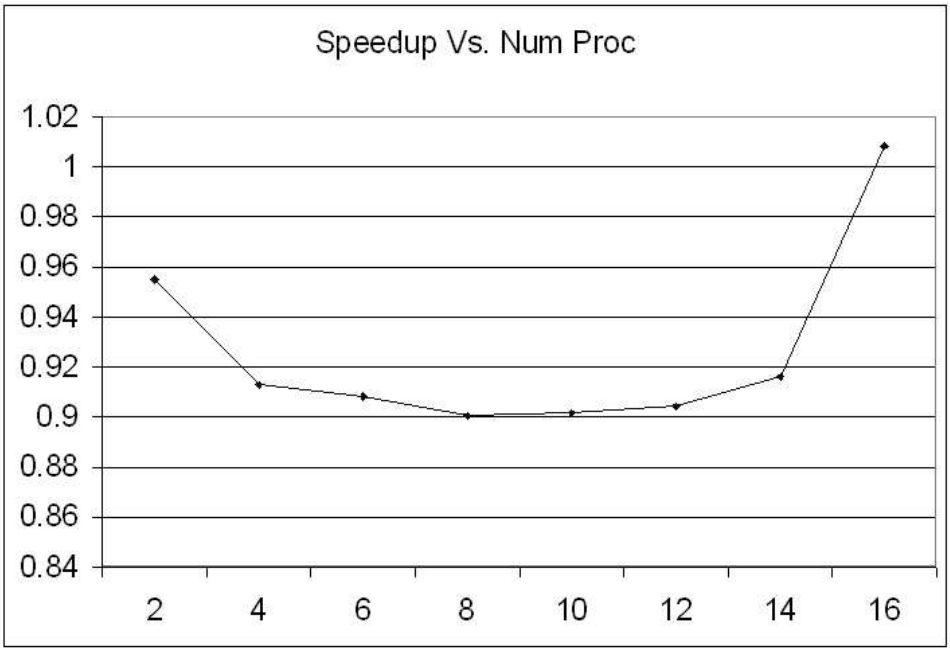

Figure 44 Parallel GA Speedup for 2-16 Processors

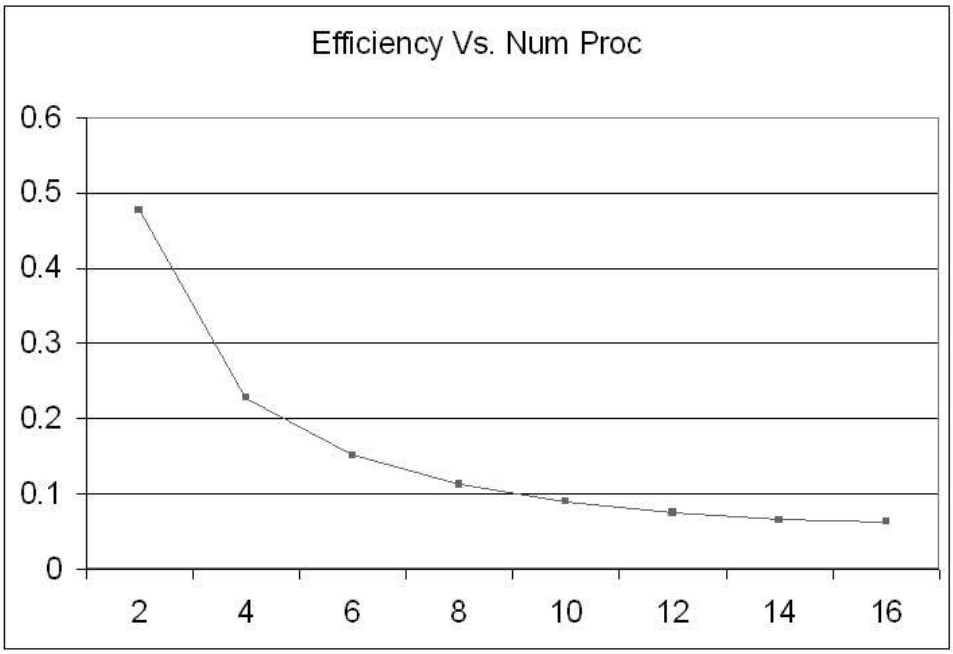

Figure 45 Parallel GA Efficiency for 2-16 Processors

sors, does not improve the overall effectiveness of the algorithm. The sharp dip in performance at 14 processors is likely due to errant variables during startup.

6.2 AIS Results & Analysis

The AIS design of experiments sought to discover good variable values to achieve a high Detection Rate (DR). This was an iterative process in which values were discovered at each step to be used in the following test. All figures presented include the average detection rate and False Alarm Rate (FAR) for different values of the given variable. If appropriate, error bars are included in the charts; however, most variance values were so low that it is not possible to visibly see the degree of variance.

6.2.1 Antibody Size. AIS Test 1 sought to determine the best antibody size for detection of the antigens described in Subsection 3.1.2.6. This was chosen as the first test due to high impact that the size of antibodies has upon system performance

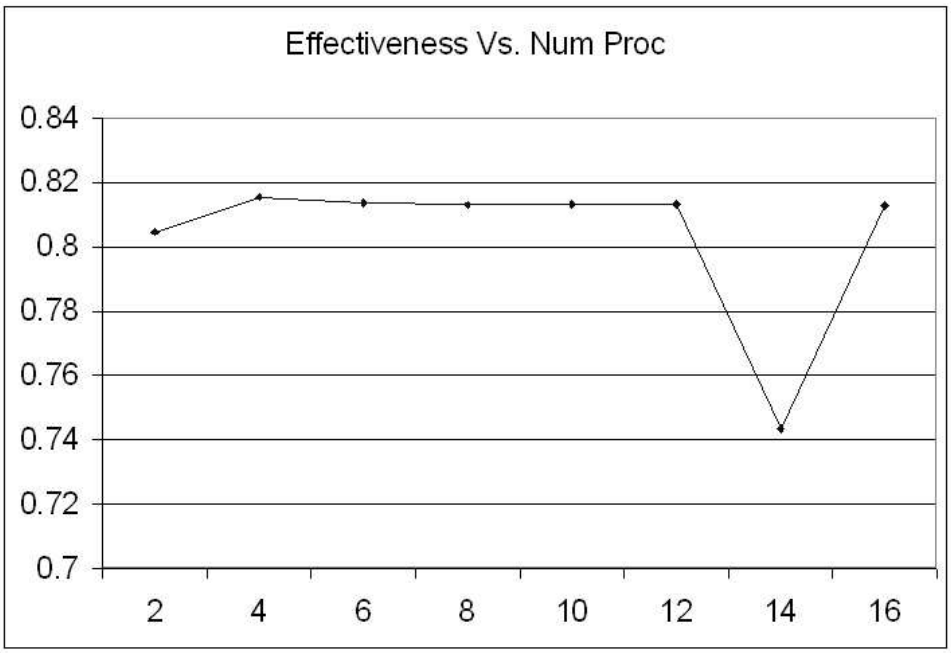

Figure 46 Parallel GA Effectiveness for 2-16 Processors

and the detection rate. An antibody that is very small relative to the signature to be classified is too general to distinguish between many of the different possible measurement signatures; while an antigen that is close to the size of the signature may be too specific to produce induce a match. An antibody with a balance between generality and specificity is ideal. Figure 47 indicates the detection rate and false alarm rate for antibodies of size 16 to 64. The measurements being classified were 64 bits in length. The figure indicates that an antibody length of 16 achieves good results and is used in Test 2. Measurement variance was very small (0.005) and is not visible in the figure.

6.2.2 Match Threshold. AIS Test 2 tested the impact of the match threshold on detection rate. The match threshold determines the point at which a sensor determines that a measurement is non-self and sends a warning to the network node. This value is the first layer of detection as the warning is then checked for costimulation at the network node. The value should be as low as possible in order to prevent

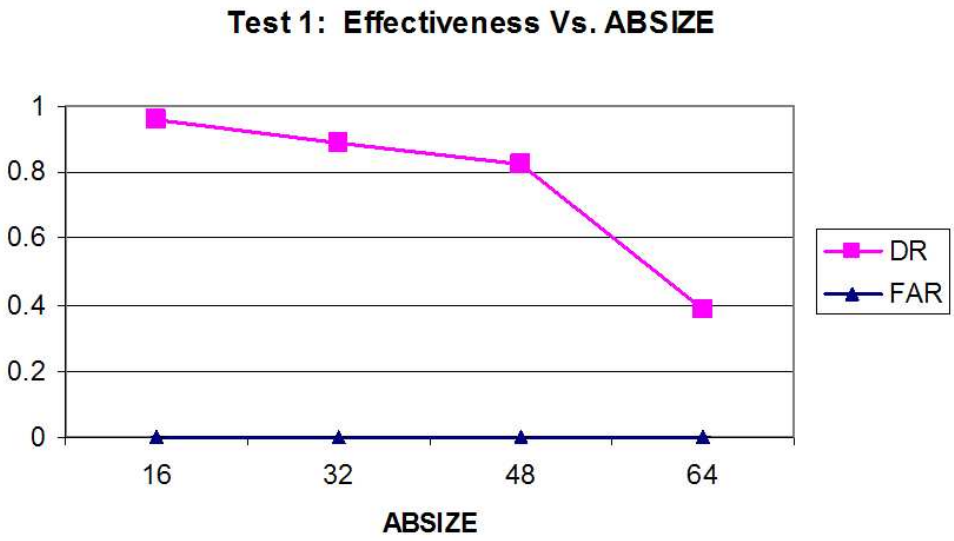

Figure 47 AIS Test 1: Antibody Size Vs. Effectiveness

a measurement that is anomalous from being classified as benign (a False Negative). Figure 48 tests values from 0.9 to 0.4. As illustrated, values higher than 0.6 result in misclassification by producing False Negatives for nearly all measurements. Values less than 0.6 do not significantly improve the detection rate. Based upon these results, a value of 0.6 is the best choice for match threshold. Values lower than 0.6 may also produce unnecessary False Positives, thereby lowering the detection rate.

6.2.3 Costimulation Threshold. The costimulation threshold sets the point at which a measurement is validated as self/non-self, producing a warning at the Network node. The costimulation process compares a current warning to warnings previously received to reduce the likelihood that a single False Positive may propagate to the Global node as a valid warning. AIS Test 3 tested values of 0.9 to 0.7 to determine the best costimulation threshold. Results are shown in Figure 49. As shown by the figure, the detection rate improves marginally as the threshold is increased. A high costimulation threshold is warranted in order to reduce the likelihood of false positives. For this reason, a costimulation threshold of 0.9 was chosen for the next 4 tests.

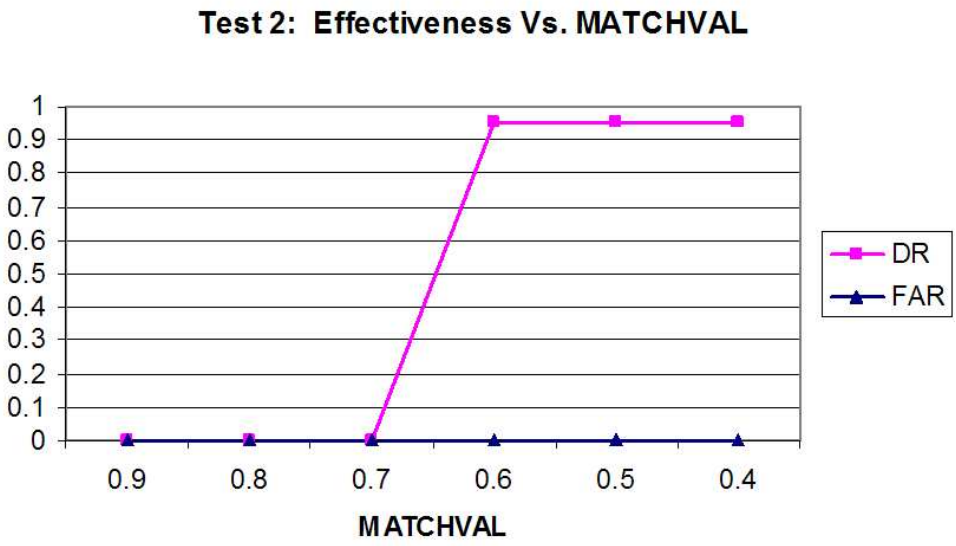

Figure 48 AIS Test 2: Match Threshold Vs. Effectiveness

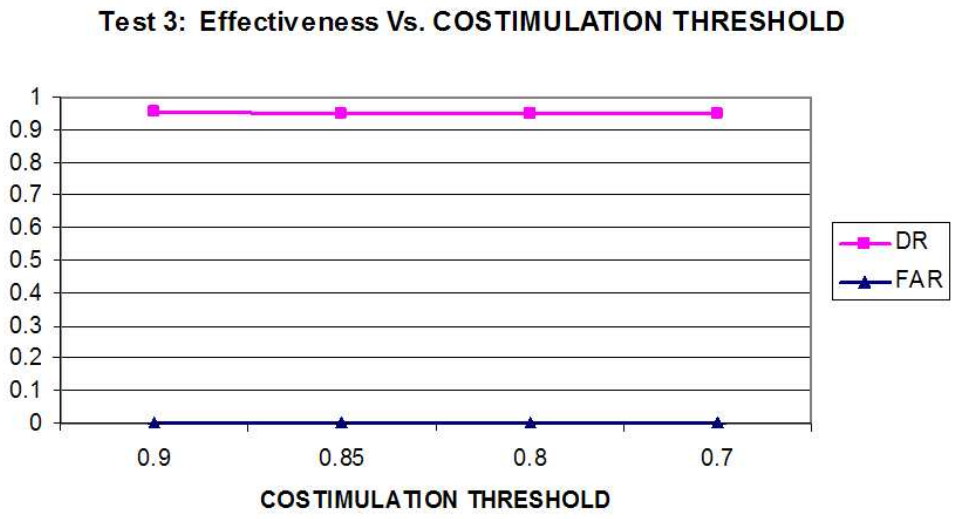

Figure 49 AIS Test 3: Costimulation Threshold Vs. Effectiveness

6.2.4 Number of Antibodies. AIS Test 4 varied the number of antibodies initially introduced via negative selection. Values of 10, 25, 50, and 100 were tested producing different values for the Detection Rate. Figure 50 shows the results of this test. As shown, all four values produced similar results. There was, however, a marginal decrease in effective rate from 25 to 100 antibodies. This is likely due to false positives produced when benign measurements manage to produce a match value high enough to exceed the threshold. The more antibodies in the system, the higher the likelihood of a match between any measurement and an antibody. Another factor to consider when choosing the number of antibodies is the impact of more antibodies on AIS performance. Due to the high algorithmic complexity of the negative selection, match, and costimulation operations, a high number of antibodies dramatically decreases system performance. Therefore, it is beneficial to choose the lowest number of antibodies necessary to achieve a high detection rate. A value of 10 is best value in this case.

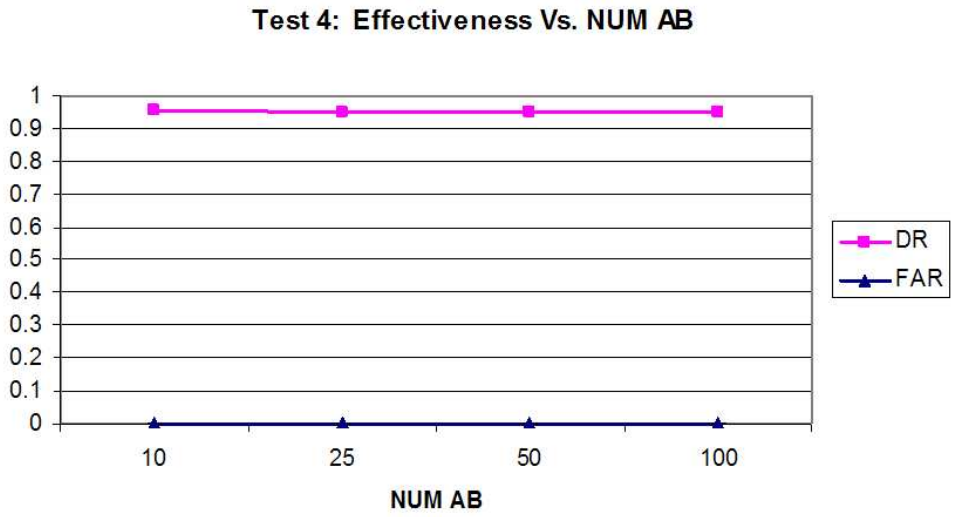

Figure 50 AIS Test 4: Number of Antibodies Vs. Effectiveness

6.2.5 Number of Self. The number of self cells impacts the AIS's ability to create antibodies quickly via negative selection. The higher the number of self,

the more difficult it is for the system create an antibody that does not match self. This directly impacts total system runtime, primarily when antibodies are generated during node initialization and during clonal selection when additional antibodies are created to match a given set of antigens. Figure 51 demonstrates the results of AIS Test 5. In this test, the number of self was varied from 500 to 4000. As expected, when there are a higher number of self cells, the system has a harder time finding good antibodies, negatively impacting the detection rate. The choice for number of self cells is arbitrary and depends upon the associated classification application. In the case of spectra recognition, a limited number of self cells would be introduced in order to represent normal and benign environmental measurements. Self measurements would likely tend to cluster in a small area and could be reasonably represented by a small sample of that area. For this reason, a value of 1000 self cells is chosen.

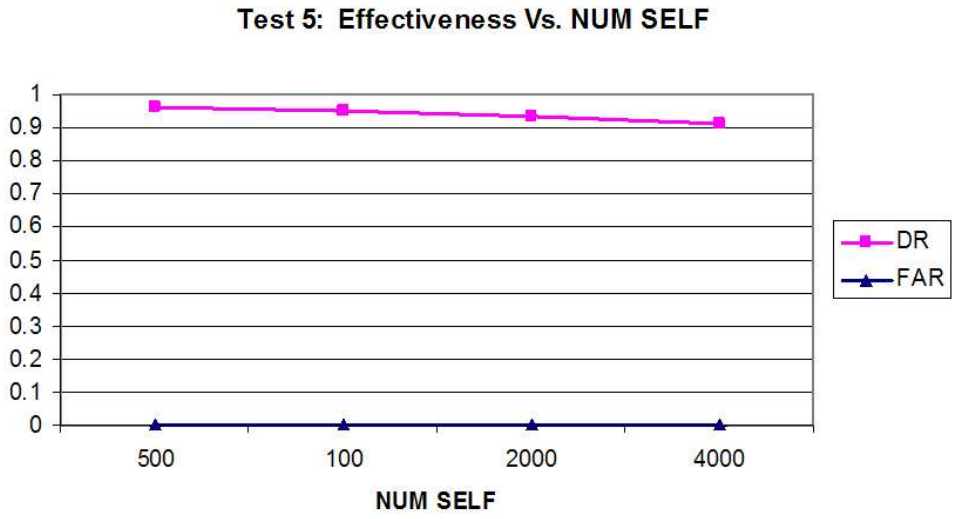

Figure 51 AIS Test 5: Number of Self Vs. Effectiveness

6.2.6 Number of Antigen Injects. The number of possible toxic chemical signatures directly effects the detection rate more than any other variable. Figure 52 demonstrates the impact of increasing the number of possible antigen measurements on detection rate. As expected, introducing additional varying antigens to a set of measurements dramatically decreases the detection rate. This is likely due to the AIS's tendency to slowly respond to new antigens, requiring time to build antibodies that detect these antigens. When only one type of antigen is introduced periodically, the system is able to quickly detect and adapt antibodies to improve detection; however, when up to 20 different antigens are randomly presented to the system, those antigens that have not yet been encountered are likely to go undetected until they are present in the system long enough for clonal selection and affinity maturation processes to produce appropriate antibodies.

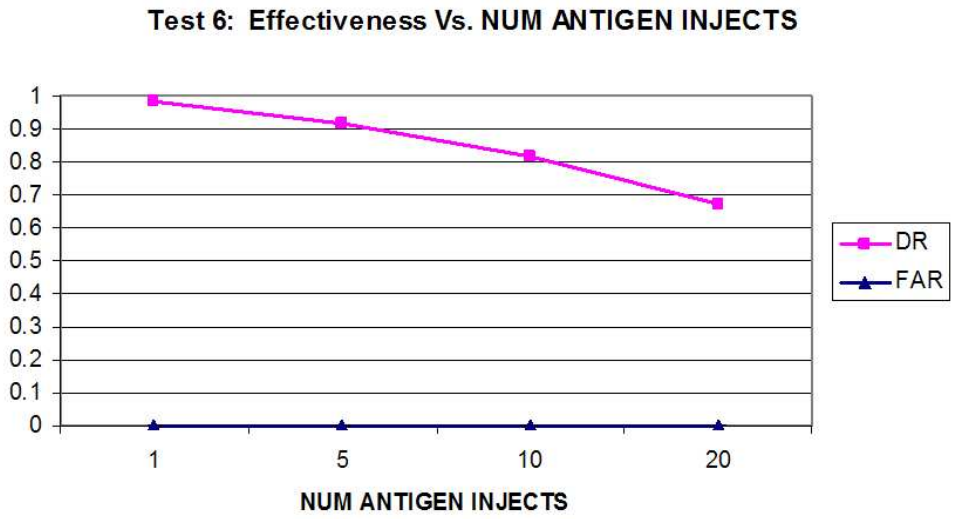

Figure 52 AIS Test 6: Number of Antigen Injects Vs. Effectiveness

6.2.7 Total Number of Measurements. The number of measurements variable determines how many measurements each sensor compares to its local antibodies in search of a match. At each time-step, the measurement operation returns either a random measurement (self) signature or a signature that represents a known

antigen. The percentage of measurements that are antigens is determined by the "ANTIGENPERCENTAGE" variable. For these tests, ANTIGENPERCENTAGE was set to 0.1; roughly 10% of the measurements returned antigen signatures. Increasing the total number of measurements introduces a higher number of anomalous measurements to the system; however, as Figure 53 indicates, the system was still able to maintain a high detection rate over a range of 500 to 5000 measurements per sensor.

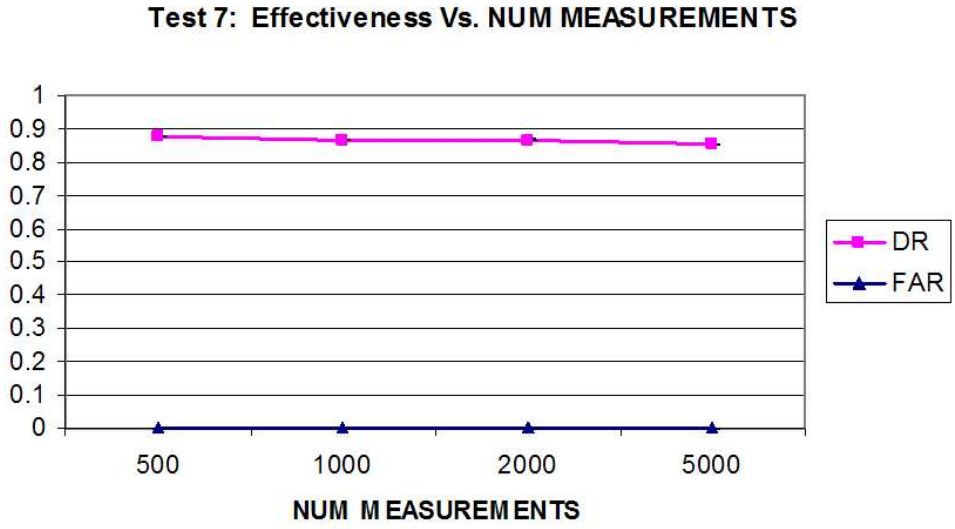

Figure 53 AIS Test 7: Number of Measurements per Sensor Vs. Effectiveness

6.2.8 Summary of AIS Results. After completion of AIS Tests 1 through 7, the following can be ascertained:

1. The best *antibody size* for detection of 64-bit signatures is 16-bits. This results in an average Detection Rate of 96%.
2. The best *match threshold* is 0.6. This results in an average Detection Rate of 95%.

3. The *costimulation threshold* may be chosen to be anywhere in a range of 0.5 to 0.9; however, in order to reduce the likelihood of false positives, a value at the higher end of this range returns good results.
4. The *number of antibodies* chosen for initialization at startup directly impacts system performance and time of execution. It is beneficial to choose a number low enough to produce a high detection rate, without introducing false negatives.
5. The *number of self* cells chosen for comparison during the negative selection operation should be as low as possible, without impacting the detection rate. A high value impacts system performance due to the direct impact on system performance. The value chosen should be representative of the actual number of self cells in the real system.
6. The *number of antigen injects* directly impacts the detection rate more than any other variable. A high number of injects significantly reduces the detection rate. This value should also be chosen to reflect the number of possible anomalous signatures that could be introduced in a real-time AIS environment. For example, if the operator wishes to only detect three different antigens, only three injects should be introduced.
7. Taking more measurements per sensor slightly lowers the overall system detection rate; however, the system still returns relatively good detection numbers.

6.3 pGRaCCE Results & Analysis

Results shown in Figure 54 and Figure 55 demonstrate the high cost of interprocessor communications during execution. Note the gradually improving trend in execution times with an increased number of processors in the 1000 generation instances versus a gradually worse time in the 10 generation instances. This is due to a low setup time (t_s) relative to the total time of execution (T_s). The improving trend in the 1000 generation instances also reflects the benefit of pGRaCCE task

decomposition by allowing more processors to focus on classifying a subset of the total data set.

Note that standard deviation is quite high in some of the Aspen runs. After checking with other system users, it was determined that other students were using Aspen at the same time, increasing processor utilization and adversely affecting execution times.

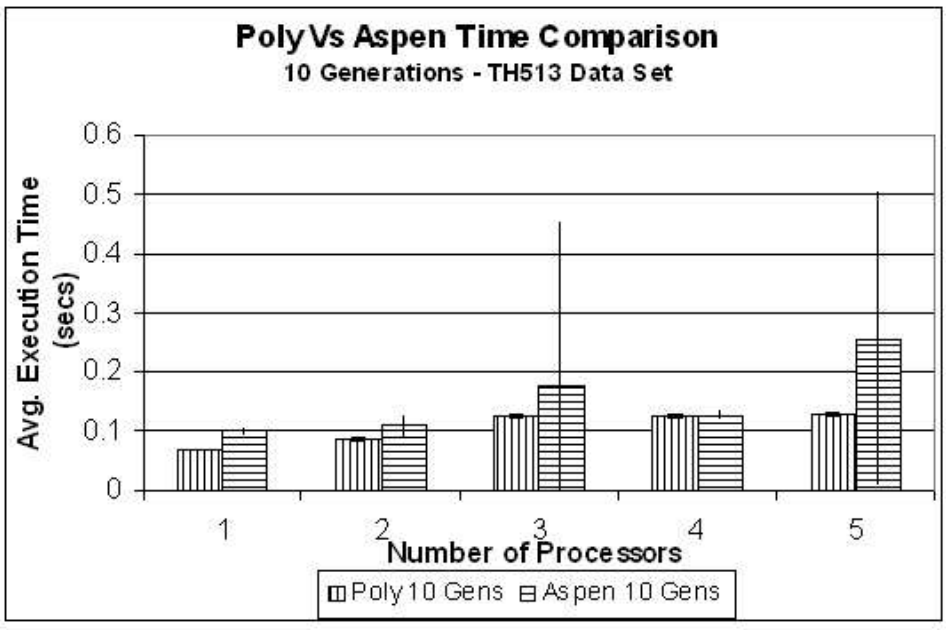

Figure 54 pGRaCCE Execution Times for 10 gen

pGRaCCE speedup results for 1000 generation tests are shown in Figure 56. The 1000 generation experiments produced gradually improving speedup, though it seemed to level out as the number of processors approached 5. In all cases, speedup was sub-linear, never really approaching the linear speedup line. The speedup results for the 10 generation tests (Figure 57) did not even approach a value of 1.0, and actually gradually declined as the number of processors increased. This was expected due to increasing execution times for that number of generations.

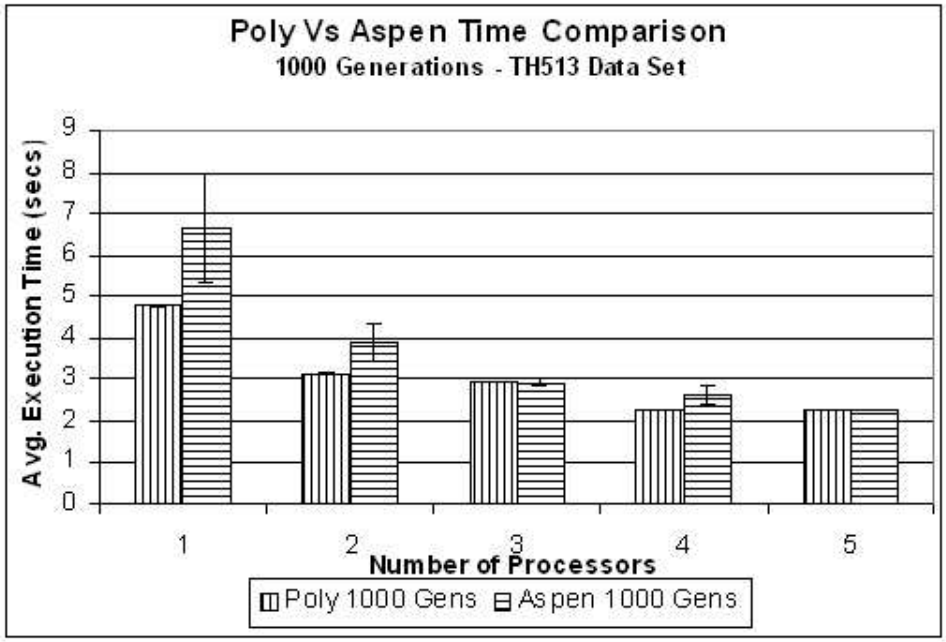

Figure 55 pGRaCCE Execution Times for 1000 gen

Finally, Figure 58 demonstrates the low efficiency of all 1000 generation experiments on Poly and Aspen. Again, the low computation times relative to communications times produced efficiencies commensurate with Figure 56 results.

6.4 Summary

As expected, the GA was able to aptly evolve antibodies capable of detecting Acetone and Methanol. Results of experiments demonstrated the ability of a GA to transform an initial random population of random antibodies to one capable of detecting the desired elements.

After determination of the best variable settings for the AIS algorithm, the system performed admirably. With no false alarms and a high detection rate, the system may be used to perform reliable self/non-self discrimination with similar data sets encoded using the schema in subsection 3.1.2.6.

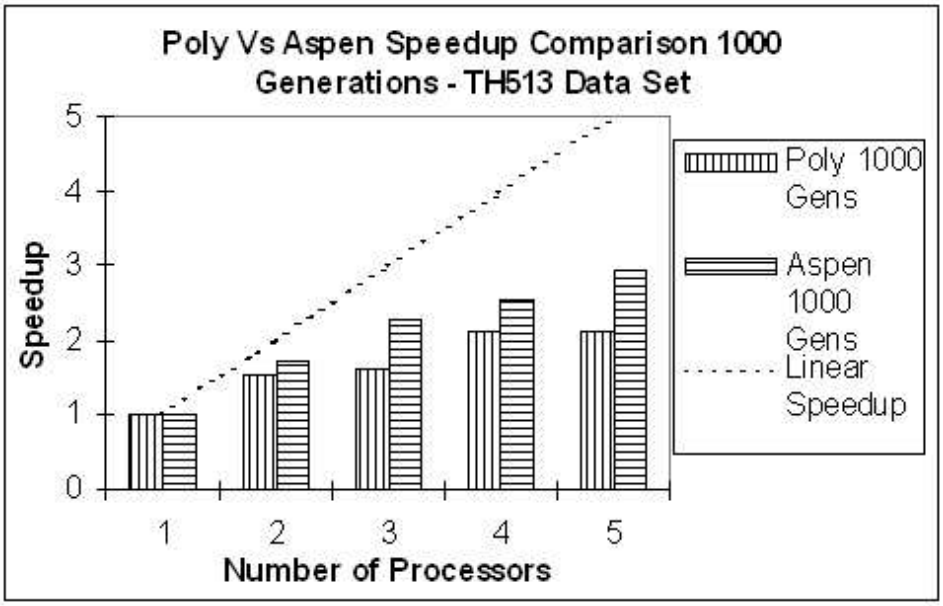

Figure 56 pGRaCCE Speedup for 1000 gen

The pGRaCCE algorithm determined the best set of features to properly classify the TH513 data set. Increasing the number of processors does improve efficiency when the serial execution time is extremely small. In this case, communications times nullified any possible benefit of parallelization. However, parallelization may be beneficial in cases of extremely large data sets with many different classes when executed to greater 1000 generations. Parallel GRaCCE has proven to benefit the proposed DAIS design. With extremely fast execution times, the algorithm may improve the ability of a real-world DAIS to detect and classify chemical spectra measurements as closely to real-time as possible. Though only the TH513 data set was used in this instance, other data sets may be easily introduced for feature extraction to enhance the classification ability of other algorithms.

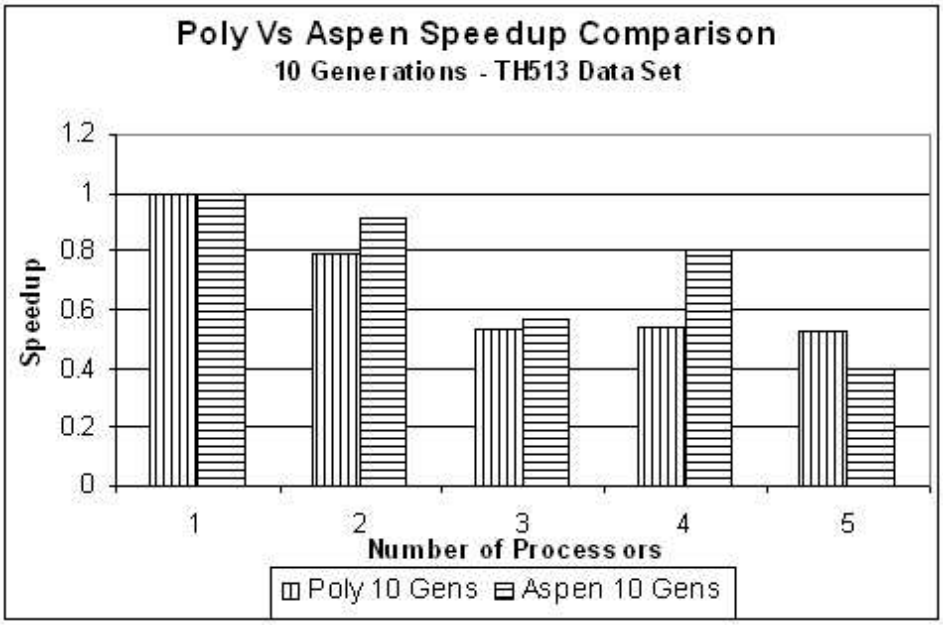

Figure 57 pGRaCCE Speedup for 10 gen

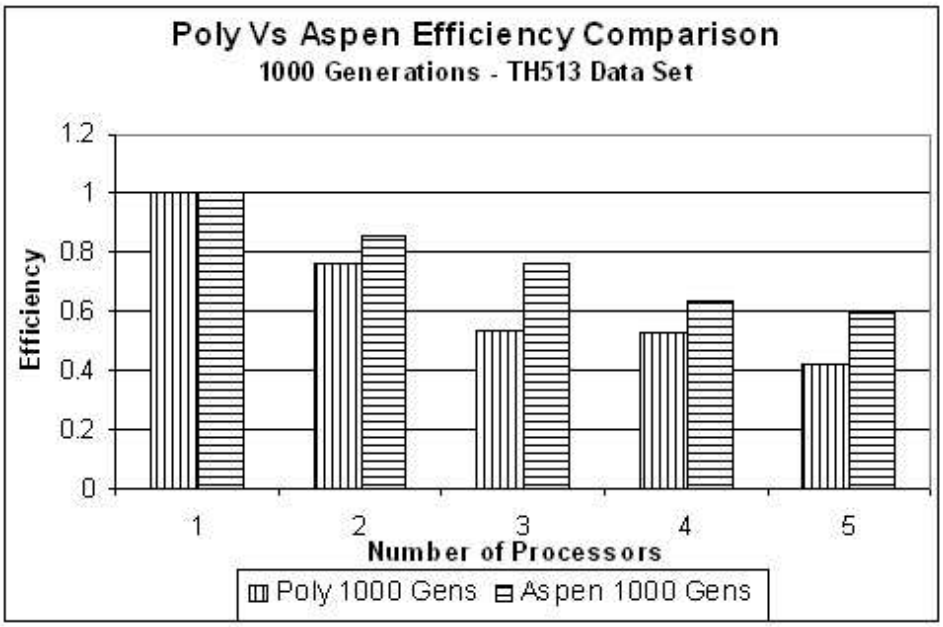

Figure 58 pGRaCCE Efficiency for 1000 gen

VII. CONCLUSIONS AND RECOMMENDATIONS

The overarching research goal was to design a distributed AIS capable of classifying anomalous measurements. This goal was successfully accomplished through the design and implementation of a system that meets the objectives established in subsection 1.2:

1. **Objective 1: Analyze the performance of pGRaCCE on a real-world data set:** The parallel performance of pGRaCCE on the th513 data set was assessed (Section 6.3) by collecting multiple parallel metrics and statistics including efficiency and speedup. pGRaCCE performed admirably, returning classification rules that could be used to discriminate between the 5 different data set classes. When run to less than 1000 generations, the speedup obtained from parallelization of GRaCCE was extremely small. However, parallelization became more beneficial when iterating over the data for greater than 1000 generations.
2. **Objective 2: Analyze the performance of a parallel implementation of Genesis:** The serial version of Genesis was successfully parallelized using MPI constructs. The parallel version was then assessed (Section 6.1.2) to determine its ability to evolve antibodies capable of classifying multiple antigens. The system proved successful in reaching this goal by evolving generalist antibodies to classify both methanol and acetone signatures with a relatively high degree of fitness. This version of Genesis could be integrated with the overall DAIS design front-end to produce high affinity antibodies and improve the overall detection rate. Further, when combined with standard AIS operations such as clonal selection and affinity maturation, even better performance may be realized.
3. **Objective 3: Design,implement, and test a basic DAIS that models a real-world network of sensors capable of classifying chemical spectra**

and producing warnings when *non-self* chemicals are present: A DAIS was designed (Section 3.1.2) and implemented (Section 4.1.1) in Java using mpiJava constructs for collective communications. The implemented system simulated a possible real-world DAIS consisting of multiple sensors that detect nearby contaminants when present. Good variables were obtained (Section 6.2) through an iterative testing process (Section 5.4) designed to focus on individual variables at each stage. The system returned an average greater than 90% detection rate with a 0% false alarm rate.

7.1 Conclusions

Genetic Algorithms. Genetic algorithms may be used to complement the development of pattern recognition systems. The biological immune system was used as a model for implementation of a system that includes operators and constructs capable of recognizing anomalous chemicals by their raw binary data signature. The proposed GA solution used ideas pioneered by Forrest, et. al. [30, 67, 31] to integrate AIS concepts with the GA domain, solving a difficult problem within the NP-complete problem domain. The simple GA was able to discover good antibodies in an extremely large search space ($O(2^n)$).

Artificial Immune Systems. The AIS model provides constructs for implementation of system capable of chemical classification. Biological operators such as clonal selection, affinity maturation, and costimulation play an integral role in the performance of the AIS. These operators in are utilized in varying capacities in an effort to simulate natural immune processes. The proposed DAIS does not always find the best solution for all similar NP-complete problems (no free lunch theorem); however, by iteratively evaluating the impact of individual AIS parameters, it is possible to identify the approach that provides good results for other problem domains.

Data Mining and Feature Subset Selection. Data mining and feature subset selection can be leveraged by using GRaCCE to reduce the dimensionality of the GA and AIS search spaces, resulting in improved performance.

Parallel Processing. Parallelization of GA and AIS operations improves performance via task decomposition results in a broader coverage of the problem domain and reduction in execution times.

7.2 Recommendations

Future work in this area may include the following areas of research:

- Real-time incorporation of pGRaCCE and Genesis into the DAIS algorithm to iteratively improve the detection rate
- Incorporation of real-world "Electronic Nose" measurement data may provide a more realistic understanding of performance in a fully implemented DAIS that uses systems on a chip (SOC) technology and wireless communications
- The addition of load balancing principles in the parallel Genesis and GRaCCE implementations to improve classification speed of large and high dimensional data sets

7.3 Summary

A strategy for the design and implementation of an AIS for robust chemical spectra classification has been presented and analyzed. This strategy incorporates concepts from many different disciplines. Evolutionary, biological, and immunological principles are mapped to the computational domain, providing the basis for genetic algorithm and artificial immune system operations. Parallel and distributed computing concepts are implemented throughout to capitalize upon the benefits of task and data decomposition. Data mining and feature subset selection principles

are also incorporated to improve system performance. The synthesis of these concepts has enabled the implementation of a distributed AIS that meets the stated research goal of robust chemical classification. Recommendations presented for future research may further improve results and enable the realization of a real-life system in accordance with the strategy presented herein. This system would require the design and fabrication of unique hardware sensors, possibly using current or future systems on a chip technology. Given the motivation of protecting civilians and military forces from becoming victims of chemical and biological warfare, the future of this technology is bright and the applications are limitless.

APPENDICES

A-1 Evolutionary Algorithms

As a component of the DAIS, genetic algorithms provide the evolutionary ability to improve system performance and classification ability. One of the first descriptions of the use of an evolutionary processes for computer problem solving appeared in articles by Friedberg in 1958 [32] and 1959 [33]. "This work represented some of the early work in machine learning and described the use of an evolutionary algorithm for *automatic programming*, i.e. the task of finding a program that calculates a given input-output function" [21]. Many studies sprung from this paper and others by Bremermann in 1962 [9], Box in 1957 [7], and Box et. al in 1969 [8]. As is the case with many ground-breaking research ideas, these early studies were reviewed with skepticism. However, by the mid-1960's the bases for the three main focuses of evolutionary computation were clearly established [21]. These three main focuses were:

Evolutionary Programming (EP): Devised by Lawrence J. Fogel in 1960 while serving at the National Science Foundation (NSF). "Fogel made the observation that intelligent behavior requires the ability of an organism to make correct predictions within its environment, while being able to translate these predictions into a suitable response for a given goal" [63]. This early work focused on evolving finite-state machines (see Mealy(1955) [57], and Moore(1957) [58]) which provided a generic test-bed for this approach.

Evolutionary Strategies (ES): Pioneered by Bienert, Rechenberg, and Schwefel at the Hermann Föttinger Institute of the Technical University of Berlin in 1964. The three students were attempting the minimize the total drag of three-dimensional slender bodies in a turbulent flow, and hit upon the idea to solve the intractable problem with the help of some kind of robot. This

"robot" would perform the necessary optimization by successively manipulating a flexible model positioned at the outlet of the wind tunnel [65]. A robot was constructed, however, it was only able to manipulate one decision variable at a time, resulting in solution stuck in local minima. A breakthrough was reached when they decided to switch to small random changes that were only accepted in the case of improvements. "The interpretation of binomially distributed changes as mutations and of the decision to step back or not as selection (on 12 June 1964) was the seed for all further developments leading to evolution strategies (ESs) as they are known today" [65].

Genetic Algorithms (GAs): First conceptualized by Holland in many of his papers written in the early 1960's (e.g. see [45]). Holland set out to understand the underlying principles of adaptive systems–systems capable of responding to interactions with their environment through self-modification. By the mid-1960's, Holland's ideas began to take computational form in thesis work of several of Holland's PhD students. The distinctive feature of these theses was the successful use of competition and innovation to provide the ability to dynamically respond to unanticipated events and changing environments.

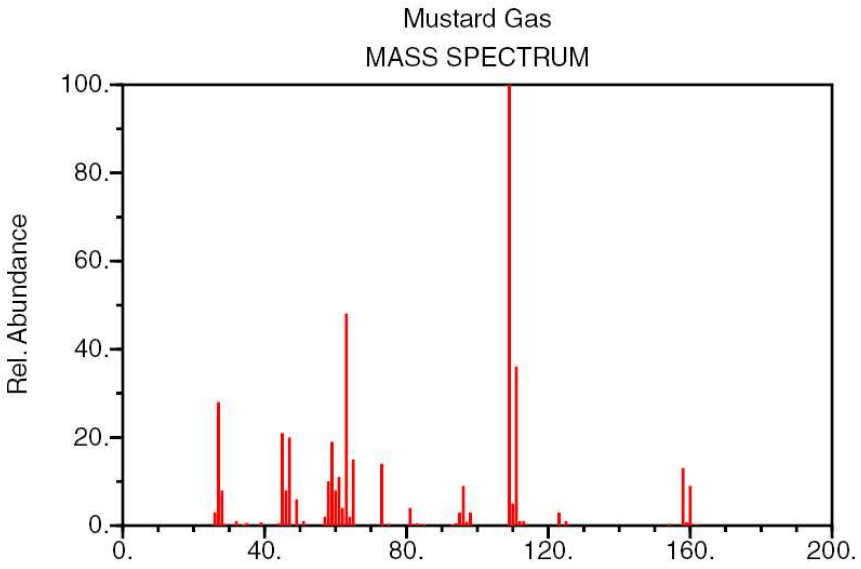

Figure 59 Mustard Gas Mass Spectra Plot [60]

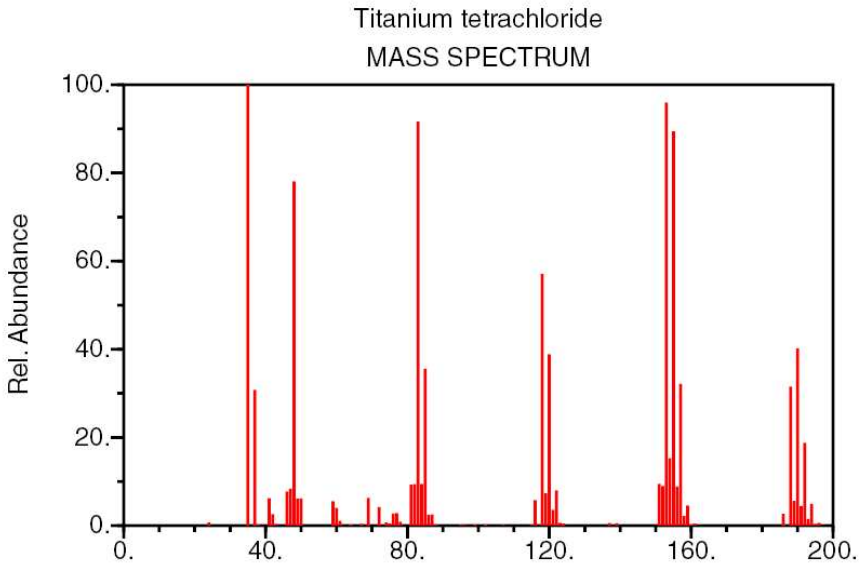

Figure 60 Titanium Tetrachloride Spectra Plot [60]

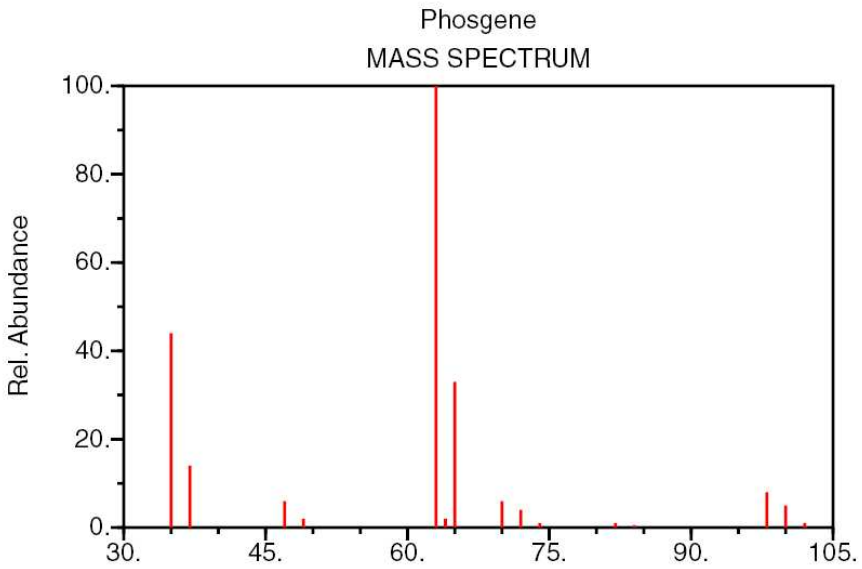

Figure 61 Phosgene Mass Spectra Plot [60]

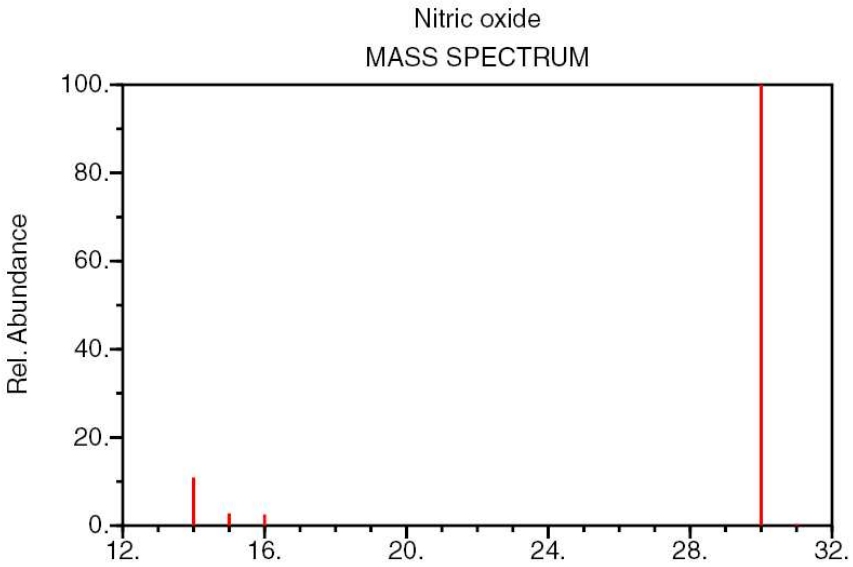

Figure 62 Nitric Oxide Mass Spectra Plot [60]

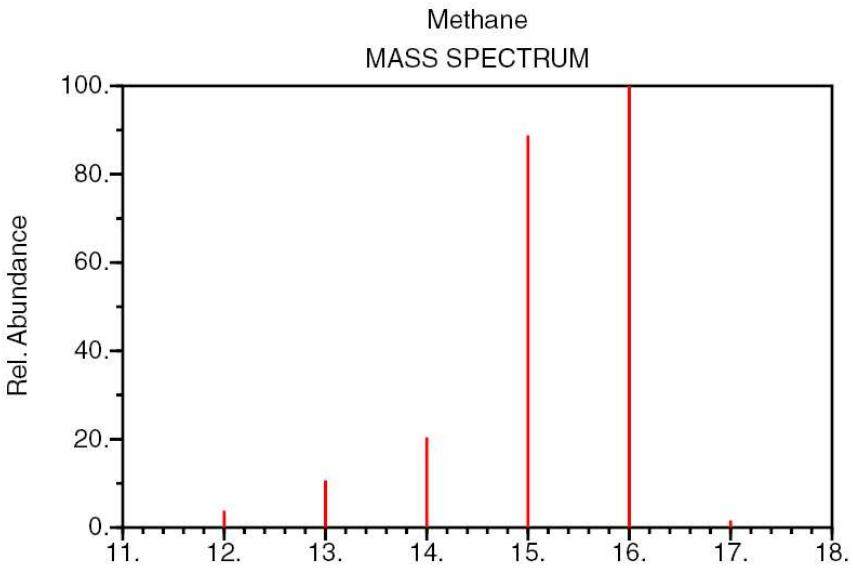

Figure 63 Methane Mass Spectra Plot [60]

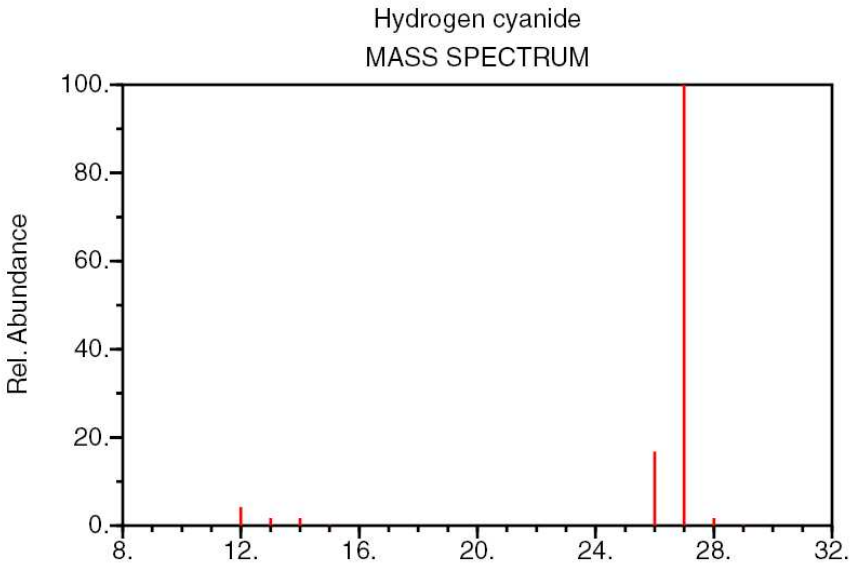

Figure 64 Hydrogen Cyanide Mass Spectra Plot [60]

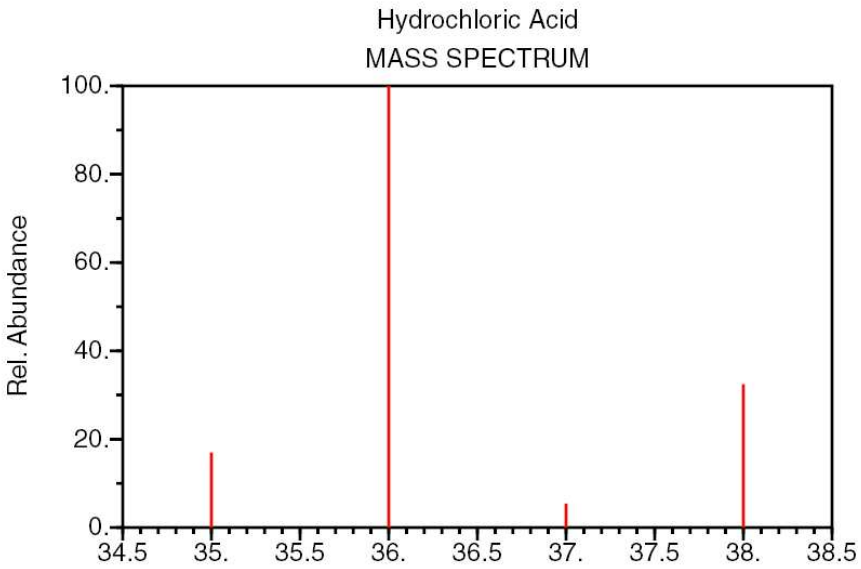

Figure 65 Hydrochloric Acid Mass Spectra Plot [60]

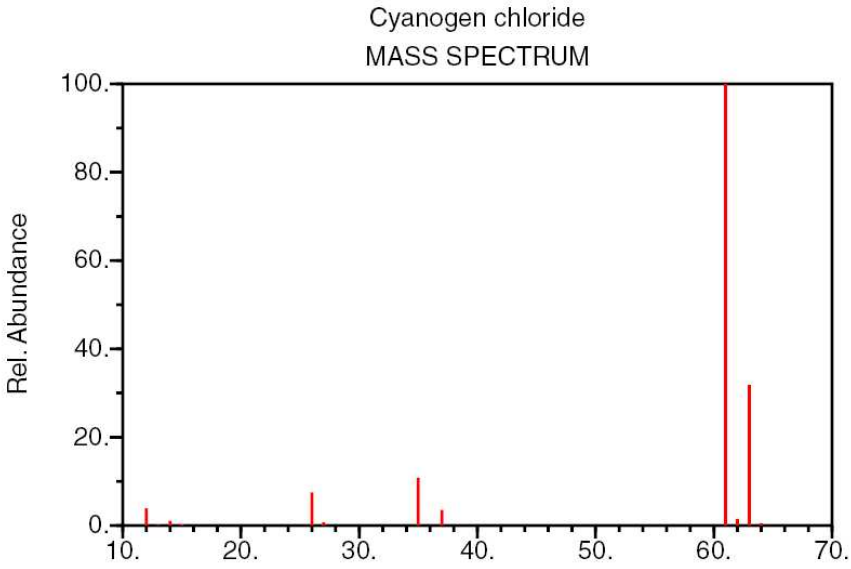

Figure 66 Cyanogen Chloride Mass Spectra Plot [60]

A

AB_CELL_STIMULATION_PERCENT - percent change in ab stimulation value

AB_EXEC_TIME - time that antibody should stay alive

AB_LENGTH - length of antibody

ACTIVATED - Static variable in class dais.cell

addCell(cell) - Method in class dais.population

addFeature(feature) - Method in class dais.cell

addFeature: add feature to current cell

addFeatureMinMax(int[]) - Method in class dais.feature_map

affinity - Variable in class dais.cell

AFFINITY_CHANGE - Static variable in class dais.ais

affinityMaturation(antibody, float, self, Random) - Method in class
    dais.antibodies

affinityMaturation: peforms aff mat.

affinityMaturation(float, self, Random) - Method in class dais.antibodies

affinityMaturation: peforms aff mat.

ais - class dais.ais.

ais() - Constructor for class dais.ais

antibodies - class dais.antibodies.
    population of antibodies used for detection

antibodies() - Constructor for class dais.antibodies
    antibodies constructor

antibodies(feature_map, int, self, Random) - Constructor for class
    dais.antibodies
    antibodies constructor .

antibody - class dais.antibody.

antibody() - Constructor for class dais.antibody

I

R

A-4 Source Code Availability

The source code for the AIS is not included as part of this document. Those interested in obtaining a copy should direct their requests to:

Dr. Gary Lamont
AFIT/ENG
BLDG 642
2950 HOBSON WAY
WRIGHT PATTERSON AFB OH 45433-7765

gary.lamont@afit.af.mil

Bibliography

1. Abdel-Aty-Zohdy, H. S., et al. "Multidisciplinary Biotechnology Domains for Physical Sensors and Detectors," *IEEE 4th European Workshop on Microelectronics Education* (May 2002).

2. Anchor, Kevin P., "The Computer Defense Immune System: Current and Future Research in Intrusion Detection," 2002.

3. Bäck, Thomas. *Evolutionary Algorithms in Theory and Practice*. New York: Oxford University Press, 1996.

4. Bäck, Thomas. "Introduction to Evolutionary Algorithms." *Evolutionary Computation 1* edited by Thomas Bäck, et al., 59–63, Bristol, New York: Institute of Physics Publishing and Oxford University Press, 2000.

5. Blickle, Tobias and Lothar Thiele. *A Comparison of Selection Schemes Used in Genetic Algorithms*. Technical Report 11, Gloriastrasse 35, 8092 Zurich, Switzerland: None, 1995.

6. Blickle, Tobias and Lothar Thiele. "A Mathematical Analysis of Tournament Selection." *Proceedings of the Sixth International Conference on Genetic Algorithms*, edited by Larry Eshelman. pp. 9–16. San Francisco, CA: Morgan Kaufmann, 1995.

7. Box, G. E. P. "Evolutionary operation: a method for increasing industrial productivity," *Appl. Stat.*, *6*:81–101 (1957).

8. Box, G. E. P. *Evolutionary Operation. A Method for Incrasing Industrial Productivity*. New York: Wiley, 1969.

9. Bremermann, H. J. "Optimization through evolution and recombination." *Self-Organizing Systems ed. M C Yovits et. al.* Washington DC, Spartan, 1962.

10. Caltech University Microelectronics Research Group. Online., February 2003. URL: http://www.micro.caltech.edu/micro/research/electronicnose.html.

11. Cantú-Paz, Erick. *Efficient and Accurate Parallel Genetic Algorithms*. Boston: Kluwer Academic Publishers, 2000.

12. Carter, Bryan, et al., "mpiJava 1.2: API Specification." Online., February 2003. Multiple Online Sources. Google Search "mpiJava".

13. Control, Centers For Disease, "Chemical Agents." Online., February 2003. URL: http://www.bt.cdc.gov/Agent/agentlistchem.asp.

14. Corradi, Antonio, et al. "Diffusive Load-Balancing Policies for Dynamic Applications," *IEEE Concurrency*, *7*(1):22 (1999).

15. Costa, A. M. "Makespan Generation on Parallel Processors: An Immune-Based Approach." *Proc. of the IEEE*. 2002.

16. Dasgupta, D., "Artificial Neural Networks and Artificial Immune Systems: Similarities and Differences," 1997.

17. Dasgupta, Dipankar. *An Overview of Artificial Immune Systems and Their Applications*, chapter 1, 3–21. New York: Springer, 1998.

18. Dasgupta, Dipankar, et al. "An Immunogenetic Approach to Spectra Recognition." *Proceedings of the Genetic and Evolutionary Computation Conference1*, edited by Wolfgang Banzhaf, et al. 149–155. Orlando, Florida, USA: Morgan Kaufmann, 13-17 1999.

19. Dasgupta, Dipankar and Stephanie Forrest, "Novelty Detection in Time Series Data Using Ideas from Immunology." Web doc, 1996.

20. de Castro, L. N. and J. Timmis. "Artificial Immune Systems: A Novel Paradigm to Pattern Recognition," *University of Kent at Canterbury* (1994).

21. De Jong, Kenneth, et al. "A history of evolutionary computation." *Evolutionary Computation 1* edited by Thomas Bck, et al., 40–51, Bristol and Philadelphia: Institute of Physics Publishing, 2000.

22. De Jong, Kenneth A. and William Spears. "Learning Concept Classification Rules using Genetic Algorithms." *Proceedings of the Twelfth International Conference on Artificial Intelligence IJCAI-91 2*. 1991.

23. Deb, Kalyanmoy. *Evolutionary Computation 1*, *1*, chapter 22, 166–171. Philadelphia: Institute of Physics Publishing, 2000.

24. Eshelman, Larry J. *Genetic Algorithms*, *1*, chapter 8, 64–78. Philadelphia: Institute of Physics Publishing, 2000.

25. Ewing, Dr. Robert L., "Bio-Inspired Cognitive Neural Computer for Olfactory Decision Science," 2003.

26. Farmer, J. D., et al. "The immune system, adaptation, and machine learning," *Physica*, *22D*:pp. 187–204 (1986).

27. Forrest, Stephane, et al. "Computer immunology," *Communications of the ACM*, *40*(10):88–96 (1997).

28. Forrest, Stephanie and Steven A. Hofmeyr, "John Holland's Invisible Hand: An Artificial Immune System." Presented at the Festschrift held in honor of John Holland, 1999.

29. Forrest, Stephanie, et al. "A Sense of Self for Unix Processes." *Proceedinges of the 1996 IEEE Symposium on Research in Security and Privacy*. 120–128. IEEE Computer Society Press, 1996.

30. Forrest, Stephanie, et al. "Self-Nonself Discrimination in a Computer." *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*. Los Alamitos, CA: IEEE Computer Society Press, 1994.

31. Forrest, Stephanie, et al. "Using Genetic Algorithms to Explore Pattern Recognition in the Immune System," *Evolutionary Computation*, *1*(3):191–211 (1993).

32. Friedberg, R. M. "A learning machine: part I," *IBM*, *2*:2–13 (1958).

33. Friedberg, R. M., et al. "A learning machine: part II," *IBM*, *3*:282–7 (1959).

34. Gonzalez, F. and D. Dasgupta. "Neuro-Immune and SOM-Based Approaches: A Comparison." *Proceedings of 1st International Conference on Artificial Immune Systems (ICARIS-2002)*. 9th-11th September, 2002 2002.

35. Gonzalez, Fabio, et al., "Combining Negative Selection Techniques for Anomaly Detection," 2002.

36. Grefenstette, J. J. "GENESIS: a system for using genetic search procedures.," *Proc. Conf. Intelligent Systems and Machines*, 161–165 (1984).

37. Grefenstette, John. *Evolutionary Computation 1*, *1*, chapter 23, 172 – 180. Philadelphia: Institute of Physics Publishing, 2000.

38. Haake, S. J. and E. A. Patterson. "Photoelastic Analysis of Frozen Stressed Specimens Using Spectral-contents Analysis," *Experimental Mechanics*, *32*(4):266–262 (1992).

39. Hammack, Lonnie P. *Parallel Data Mining With The Message Passing Interface Standard On Clusters Of Personal Computers*. MS thesis, Air Force Institute of Technology, Wright Patterson AFB, OH, 1999. DTIC ADA361637.

40. Harmer, Paul K. *A Distributed Agent Architecture for a Computer Virus Immune System*. MS thesis, Air Force Institute of Technology, Wright Patterson Air Force Base, OH, 2000. DTIC ADA380212.

41. Hayes, A.T., et al. "Distributed Odor Source Localization." *EEE Sensors Journal I, Special Issue on Electronic Nose Technologies2*. 260–217. 2002.

42. Hofmeyr, S., "A Immunological Model of Distributed Detection and its Application to Computer Security," 1999.

43. Hofmeyr, Steven A., "Webpage: An Overview of the Immune System," 2002.

44. Hofmeyr, Steven A. and Stephanie Forrest. "Architecture for an Artificial Immune System," *Evolutionary Computation*, *8*(4):443–473 (2000).

45. Holland, J. H. "Outline for a logical theory of adaptive systems," *Journal of the ACM*, *9*:297–314 (1962).

46. Infosphere.com, "Mass Spectroscopy." Online., February 2003. URL: http://www.informationsphere.com/pgs/body.php3?id=2176.

47. Inman, J. K. "The antibody combining region: Speculations on the hypothesis of general multispecificity," *Theoretical Immunology*, 243278 (1978).

48. Janssen, M. A., "An immune system perspective on ecosystem management.." Online., June 2002. URL: http://www.consecol.org/vol5/iss1/art13.

49. Kim, Jungwon and Peter J. Bentley. "An Evaluation of Negative Selection in an Artificial Immune System for Network Intrusion Detection." *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, edited by Lee Spector, et al. 1330–1337. San Francisco, California, USA: Morgan Kaufmann, 7-11 2001.

50. Kim, Jungwon and Peter J. Bentley, "Towards an Artificial Immune System for Network Intrusion Detection: An Investigation of Dynamic Clonal Selection," 2002.

51. Koza, John R., "Introduction to Genetic Algorithms."

52. Kumar, Vipin, et al. *Introduction to Parallel Computing Design and Analysis of Algorithms*. Redwood City, CA: The Benjamin/Cummings Publishing Company, Inc., 1994.

53. Lamont, Gary B., 2003. Course Material, CSCE 686 Advanced Algorithm Design, Air Force Institute of Technology.

54. Lamont, Gary B., et al. "A Distributed Architecture for a Self-Adaptive Computer Virus Immune System." *New Ideas in Optimization* edited by David Corne, et al., 167–183, London: McGraw-Hill, 1999.

55. Mahfoud, Samir W. *Evolutionary Computation 1*, *1*, chapter 26, 172 – 180. Philadelphia: Institute of Physics Publishing, 2000.

56. Marmelstein, Robert E. *Evolving Compact Decision Rule Sets*. PhD dissertation, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, May 1999. DTIC ADA364239.

57. Mealy, G. H. "A method of synthesizing sequential circuits," *Bell Syst. Tech. J.*, *34*:1051–79 (1955).

58. Moore, E. F. "Gedanken-experiments on sequential machines: automata studies," *Annals of Mathematical Studies*, *34*:129–53 (1957).

59. Nadler, Morton and Eric P. Smith. *Pattern Recognition Engineering*. New York: John Wiley & Sons Inc., 1993.

60. National Insitute of Science and Technology, "NIST Chemical Webbook." Online., February 2003. URL: http://webbook.nist.gov/chemistry/.

61. Ngo, J. Thomas and Joe Marks. "Spacetime Constraints Revisited," *Computer Graphics*, *27*(Annual Conference Series):343–350 (1993).

62. Pacey, M.N., et al. "The application of evolutionary and maximum entropy algorithms to photoelastic spectral analysis," *Experimental Mechanics*, *39*(4):265–273 (1999).

63. Porto, V. William. *Evoluationary Computation 1*, *1*, chapter 10, 89–102. Philadelphia: Institute of Physics Publishing, 2000.

64. Redner, A. S. "Photoelastic Measurements by Means of Computer-assisted Spectral-contents Analysis," *Experimental Mechanics*, *25*(1):148–153 (1985).

65. Rudolph, Günter. *Evoluationary Strategies*, *1*, chapter 9, 81–88. Philadelphia: Institute of Physics Publishing, 2000.

66. Sanford, R. J. and V. Iyengar. "The Measurement of the Complete Photoelastic Fringe Order Using a Spectral Scanner," *Proceedings of the Society for Experimental Mechanics Spring Conference on Experimental Mechanics*, 160–185 (1985).

67. Somayaji, Anil, et al. "Principles of a Computer Immune System." *Meeting on New Security Paradigms, 23-26 Sept. 1997, Langdale, UK*. 75–82. New York, NY, USA : ACM, 1998.

68. Strong, David. *Implementation and Analysis of the Parallel Genetic Rule and Classifier Construction Environment*. MS thesis, Air Force Institute of Technology, Wright Patterson AFB, OH, 2001. AFIT/GCS/ENG/01M-14.

69. Swets, Daniel L., et al. "Genetic algorithms for object recognition in a complex scene." *ICIP*. 2595–2598. 1995.

70. Thierens, Dirk. "Dimensional Analysis of Allele-Wise Mixing Revisited." *Parallel Problem Solving from Nature – PPSN IV*, edited by Hans-Michael Voigt, et al. 255–265. Berlin: Springer, 1996.

71. Van Veldhuizen, David A., et al. "Consideration in Engineering Parallel Multiobjective Evolutionary Algorithms." Unpublished as of June 2002, to be published in IEEE Transactions on Evolutionary Computation, 2002.

72. Voloshin, A. S. and A. S. Redner. "Automated Measurment of Birefringence: Development and Experimental Evaluation of the Techniques," *Experimental Mechanics*, *29*(1):252–257 (1989).

73. Whitley, Darrell. "A Genetic Algorithm Tutorial," *Statistics and Computing*, *4*:65–85 (1994).

74. Yilmaz, Okan. *Data Mining Feature Subset Weighting and Selection Using Genetic Algorithms*. MS thesis, Air Force Institute of Technology, Wright Patterson AFB, OH, 2002. AFIT/GCE/ENG/02M-05.

Vita

Captain Mark A. Esslinger graduated from Faith Lutheran High School in Las Vegas, Nevada, in June 1994. He entered undergraduate studies at the United States Air Force Academy (USAFA) where he graduated with an a Bachelor or Science degree in Computer Science in May 1998. After graduation from USAFA, he entered Basic Communications Officer Training in July 1998 at Keesler Air Force Base, Mississippi, and graduated in October 1998 as a Communication Officer. His first assignment was to the 18th Communications Squadron, Kadena Air Base, Okinawa, Japan, as the Wing Information Assurance Officer and later as Officer in Charge of the base Network Control Center. In August 2001, he entered the School of Engineering and Management at the Air Force Institute of Technology. Upon graduation, Captain Esslinger will be assigned to the Air Force Technical Applications Center (AFTAC) at Patrick Air Force Base, Florida.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
25-03-2003	**Master's Thesis**	Jun 2002 – Mar 2003

4. TITLE AND SUBTITLE

AN ARTIFICIAL IMMUNE SYSTEM STRATEGY FOR ROBUST CHEMICAL SPECTRA CLASSIFICATION VIA DISTRIBUTED HETEROGENEOUS SENSORS

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Esslinger, Mark, Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/03-06

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFTA
Attn: Dr. Robert Ewing
Information Directorate DSN: 785-6653 #3592
WPAFB OH 45433-7765 e-mail: Robert.Ewing@wpafb.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The timely detection and classification of chemical and biological agents in a wartime environment is a critical component of force protection in hostile areas. Moreover, the possibility of toxic agent use in heavily populated civilian areas has risen dramatically in recent months. This thesis effort proposes a strategy for identifying such agents via distributed sensors in an Artificial Immune System (AIS) network. The system may be used to complement "electronic" nose ("E-nose") research being conducted in part by the Air Force Research Laboratory Sensors Directorate. In addition, the proposed strategy may facilitate fulfillment of a recent mandate by the President of the United States to the Office of Homeland Defense for the provision of a system that protects civilian populations from chemical and biological agents. The proposed system is composed of networked sensors and nodes, communicating via wireless or wired connections. Measurements are continually taken via dispersed, redundant, and heterogeneous sensors strategically placed in high threat areas. These sensors continually measure and classify air or liquid samples, alerting personnel when toxic agents are detected. Detection is based upon the Biological Immune System (BIS) model of antigens and antibodies, and alerts are generated when an a measured sample is determined to be a valid toxic agent (antigen). Agent signatures (antibodies) are continually distributed throughout the system to adapt to changes in the environment or to new antigens. Antibody features are determined via data mining techniques in order to improve system performance and classification capabilities. Genetic algorithms (GA's) are a critical part of the process, namely in antibody generation and feature subset selection calculations. Demonstrated results validate the utility of the proposed distributed AIS model for robust chemical spectra recognition.

15. SUBJECT TERMS

Genetic, Algorithms, Evolution, Parallel Processors, Parallel Processing, Sensors, Detectors, Chemical Analysis, Chemical Agents, Chemical Determination, Biological Agents, Biological Warfare, Collective Protection, Data Mining, Feature Extraction, Classification, Distributed Data Processing, Distributed Interactive Simulation, Spectra

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Gary B. Lamont, Professor, CIV, AFIT/ENG
U	**U**	**U**	**UU**	162	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636, ext 4718; e-mail: gary.lamont@afit.edu